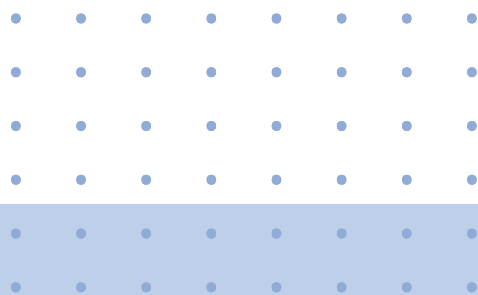


**Microsoft®**

# **Guia de Arquitetura e Projeto de Smart Clients**



**Prefácio de Mark Boulter**

patterns & practices

As informações contidas neste documento, incluindo referência URL a outros sites da internet, estão sujeitas a alteração sem aviso prévio. A menos que seja indicado o contrário, as empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, locais e eventos aqui citados são fictícios, e nenhuma associação a qualquer empresa, organização, produto, nome de domínio, endereço de e-mail, logotipo, pessoa, lugar ou eventos reais é intencional ou deve assim ser inferida. Atender a todas as leis de direitos autorais aplicáveis é responsabilidade do usuário. Sem limitar os direitos autorais, nenhuma parte deste documento deve ser reproduzida, armazenada ou introduzida em um sistema de recuperação, ou transmitida de qualquer forma ou por qualquer meio (eletrônico, mecânico, fotocópia, gravação ou outros) ou para qualquer finalidade sem o consentimento expresso por escrito da Microsoft Corporation.

A Microsoft pode possuir patentes, solicitações de patentes, marcas comerciais, direitos autorais ou outros direitos de propriedade intelectual cobrindo itens citados neste documento. Exceto se expressamente indicado em qualquer acordo de licença por escrito da Microsoft, o fornecimento deste documento não concede nenhuma licença a essas patentes, marcas comerciais, direitos autorais ou outra propriedade intelectual.

© 2006 Microsoft Corporation. Todos os direitos reservados.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Active Directory, BizTalk, InfoPath, MSDN, Outlook, Visual Basic, Visual C++, Visual C#, Visual Studio e Win32 são marcas comerciais da Microsoft Corporation nos Estados Unidos e/ou em outros países.

Todas as outras marcas comerciais são de posse de seus respectivos proprietários.

# **Guia de Arquitetura e Projeto de Smart Clients**

patterns & practices

David Hill, Microsoft Corporation.

Brenton Webster, Microsoft Corporation.

Edward A. Jezierski, Microsoft Corporation.

Srinath Vasireddy, Microsoft Corporation.

Mo Al-Sabt, Microsoft Corporation.

Blaine Wastell, Ascentium Corporation.

Jonathan Rasmusson, Thoughtworks.

Paul Gale, ThoughtWorks.

Paul Slater, Wadeware LLC.

# Guia de Arquitetura e Projeto de Smart Clients

## Índice

Prefácio .....	vii
----------------	-----

## Capítulo 1

<b>Introdução .....</b>	<b>1</b>
<b>O que é um Smart Client? .....</b>	<b>1</b>
Aplicações Rich Client .....	2
Aplicações Thin Client .....	2
Aplicações Smart Client .....	3
<b>Tipos de Smart Client .....</b>	<b>7</b>
Aplicações Smart Client do Windows .....	8
Aplicações Smart Client do Office .....	8
Aplicações Móveis para Smart Client .....	10
<b>Escolhendo entre Smart Client e Thin Client .....</b>	<b>10</b>
<b>Desafios da Arquitetura Smart Client .....</b>	<b>11</b>
<b>Escopo deste Guia .....</b>	<b>13</b>
<b>Como utilizar este Guia .....</b>	<b>13</b>
<b>Quem deve ler este Guia .....</b>	<b>14</b>
Pré-requisitos .....	14
<b>Sinopse de Capítulos .....</b>	<b>14</b>
Capítulo 1: Introdução .....	14
Capítulo 2: Manipulação de Dados .....	15
Capítulo 3: Conectando-se .....	15
Capítulo 4: Smart Clients Conectados Ocasionalmente .....	15
Capítulo 5: Considerações de Segurança .....	15
Capítulo 6: Utilizando Threads Múltiplos .....	15
Capítulo 7: Implementando e Atualizando as Aplicações Smart Client .....	16
Capítulo 8: Desempenho da Aplicação Smart Client .....	16
<b>Sumário .....</b>	<b>16</b>
<b>Mais Informações .....</b>	<b>16</b>

## Capítulo 2

<b>Manipulação de Dados.</b> . . . . .	<b>7</b>
<b>Tipos de Dados</b> . . . . .	<b>18</b>
Dados de Referência somente para leitura . . . . .	18
Dados Temporários . . . . .	19
<b>Dados em Cachê</b> . . . . .	<b>19</b>
Bloco de Aplicação em Cachê . . . . .	22
<b>Dados Simultâneos.</b> . . . . .	<b>24</b>
<b>Utilizando o DataSets do ADO.NET para Gerenciar Dados</b> . . . . .	<b>25</b>
Mesclando os Dados com os Datasets . . . . .	26
Aumentando o desempenho dos Datasets . . . . .	26
<b>Vinculação de Dados com Formulários do Window</b> . . . . .	<b>27</b>
Arquitetura de Vinculação de Dados com formulários do Windows . . . . .	28
Vinculação de dados para controles de Formulários do Windows . . . . .	30
<b>Sumário</b> . . . . .	<b>37</b>

## Capítulo 3

<b>Conectando-se</b> . . . . .	<b>39</b>
<b>Sistemas Loosely Coupled e Tightly Coupled</b> . . . . .	<b>39</b>
<b>Opções de Comunicação</b> . . . . .	<b>40</b>
Serviços.NET para Empresas . . . . .	40
.NET Remoting . . . . .	42
Message Queuing . . . . .	44
Web Services . . . . .	45
<b>Escolhendo uma Opção de Comunicação</b> . . . . .	<b>47</b>
<b>Projetando Aplicações Smart Client Conectadas</b> . . . . .	<b>48</b>
Use Mensagens Coarse-Grained, Em Cápsula . . . . .	48
Evitando Transações ACID Distribuídas . . . . .	48
Evitando o Envio de Datasets Através da Rede . . . . .	49
Dividindo Grandes Datasets . . . . .	49
Versione seus Web services e Assemblies . . . . .	49
<b>Sumário</b> . . . . .	<b>50</b>

## Capítulo 4

<b>Smart Clients Ocasionalmente Conectados</b> .....	51
<b>Cenários Comuns Conectados Ocasionalmente</b> .....	52
<b>Estratégias de Projeto Conectados Ocasionalmente</b> .....	53
Abordagem Centrada em dados .....	55
A Abordagem Orientada Para o Serviço .....	57
<b>Desenvolvimento de Aplicações Smart Client Ocasionalmente Conectadas</b>	
Utilização de Abordagem Orientada Para o Serviço .....	59
Favorecimento de Comunicação Assíncronica .....	59
Minimização de Interações Complexas de Rede .....	60
Adição de Capacidades de Cachê de Dados .....	61
Manipulando Conexões .....	62
Desenvolvimento de Mecanismo Store-and-Forward .....	64
Gerenciamento de Conflitos de Regras de negócio e Dados .....	65
Interação com Web Service CRUD .....	71
<b>Usando uma Abordagem Baseada em tarefas</b> .....	72
<b>Manipulação de Dependências</b> .....	73
<b>Sumário</b> .....	77

## Capítulo 5

<b>Considerações de Segurança</b> .....	79
<b>Autenticação</b> .....	80
Cenários de Autenticação Smart Client .....	80
Escolhendo o Modelo de Autenticação Correto .....	83
Tipos de Autorização de Acesso de Rede .....	84
Recolher e validar Credenciais de Usuários .....	89
Diretrizes de Autenticação .....	91
<b>Autorização</b> .....	92
Tipos de Autorização .....	92
Adição de Capacidades em Sua Aplicação .....	94
Diretrizes de Autenticação .....	95
Autorização de Funcionalidade Quando o Cliente está Offline .....	96
O Bloco de Autorização Aplicação e Perfil .....	97
<b>Validação de Entrada</b> .....	97

<b>Manipulação de Dados Sensíveis . . . . .</b>	<b>98</b>
Determinar Quais dados Armazenar no Cliente . . . . .	99
Técnicas para Proteção de Dados Sensíveis . . . . .	100
<b>Segurança do Acesso ao Código . . . . .</b>	<b>102</b>
Resolução de Permissão de Segurança do Acesso ao Código . . . . .	104
Desenvolvimento para Segurança do Acesso ao Código . . . . .	105
<b>Sumário . . . . .</b>	<b>112</b>

## Capítulo 6

<b>Utilizando Threads Múltiplos . . . . .</b>	<b>113</b>
<b>Threads Múltiplos no .NET Framework . . . . .</b>	<b>113</b>
Escolher entre Chamados Sincrônicos e Assíncrônicos . . . . .	114
Escolher entre Threads de Foreground e Background . . . . .	114
Manipulação de Protocolo e Sincronização . . . . .	115
Utilizando Timers . . . . .	116
<b>Quando Usar Threads Múltiplos . . . . .</b>	<b>117</b>
Comunicar-se Através de Uma Rede . . . . .	118
Executar Operações Locais . . . . .	118
Distinguir Tarefas de Prioridade Variante . . . . .	119
Startup da Aplicação . . . . .	119
<b>Criando e Utilizando Threads . . . . .</b>	<b>119</b>
Utilizando a Classe ThreadPool . . . . .	119
Utilizando a Classe Thread . . . . .	121
Utilizando Delegados . . . . .	122
Chamando Web Services Assincrônicamente . . . . .	125
<b>Utilizando Tarefas para Manipular Interações entre o Thread UI e Outros Threads . . . . .</b>	<b>126</b>
Definindo uma Task Class . . . . .	129
Utilizando uma Task Class . . . . .	134
<b>Sumário . . . . .</b>	<b>136</b>

## Capítulo 7

<b>Implantação e Atualização das Aplicações Smart Client . . . . .</b>	<b>137</b>
<b>Implementando o .NET Framework . . . . .</b>	<b>138</b>
Pré- Instalando o .NET Framework . . . . .	139
Instalação do .NET Framework com uma Aplicação . . . . .	139

<b>Implementando Aplicações Smart Client</b> .....	<b>140</b>
Implementação No-touch .....	141
Implementação de Update Stub em Aplicações No-touch. ....	144
Executando Códigos de um Arquivo Compartilhado .....	146
Implementação Xcopy. ....	147
Pacotes Windows Installer .....	147
<b>Escolhendo a Abordagem de Implementação Correta</b> .....	<b>149</b>
<b>Implementando Atualizações Smart Client.</b> .....	<b>151</b>
Atualizações da Implementação No-touch .....	152
Atualizações Automáticas .....	152
Atualização de um Arquivo Compartilhado .....	153
Atualizações Xcopy ....	153
Atualizações Windows Installer .....	154
<b>Escolhendo a Abordagem de Atualização Correta</b> .....	<b>154</b>
<b>Sumário</b> .....	<b>156</b>

## **Capítulo 8**

<b>Desempenho da aplicação Smart Client</b> .....	<b>157</b>
<b>Arquitetura pela Performance.</b> .....	<b>158</b>
Diretrizes para Colocação de Dados em Cachê .....	160
Diretrizes para Comunicação de Rede. ....	161
Diretrizes de Threading. ....	161
Diretrizes de Transações .....	163
Otimização do Tempo de Startup da Aplicação .....	164
Manipulação de Recursos Disponíveis. ....	165
Otimização de Performance do Windows Forms .....	169
<b>Ajuste e Diagnóstico de Performance</b> .....	<b>175</b>
Ajuste de Objetivos de Performance .....	175
Processo de Ajuste de Performance .....	177
Ferramentas de Performance. ....	179
<b>Sumário</b> .....	<b>183</b>
<b>Referências</b> .....	<b>183</b>
<b>Colaboradores e Revisores</b> .....	<b>185</b>
<b>Índice</b> .....	<b>187</b>
<b>Recursos Adicionais</b> .....	<b>198</b>



# Prefácio

O Microsoft®.NET Framework e o Windows Forms são grandes plataformas para a construção de aplicações Smart Client que combinam todo o poder, a flexibilidade e a grande experiência de usuário do modelo de aplicação Rich Client (Clientes Ricos) com a facilidade de implementação e estabilidade de aplicações com base em navegadores. O .NET Framework resolve conflitos de versão DLL e simplifica a implementação. O Windows Forms tem uma poderosa biblioteca de componentes de interface com o usuário e um desenvolvedor de formulários fácil de usar que combina a simplicidade de utilização do modelo de programação Microsoft Visual Basic® 6.0 com o poder e a flexibilidade do .NET Framework.

Entretanto, apesar da facilidade que o Windows Form oferece para a construção de sua interface com o usuário, ainda existem inúmeros desafios de projeto que você precisará resolver durante o desenvolvimento de sua aplicação Smart Client. Qual modelo de implementação é o correto para sua aplicação? Como permitir o processamento offline? E a segurança de dados? Como manter a aplicação com boa resposta ao usuário quando conectado com banda estreita? O que é preciso para se construir uma aplicação que vá de encontro às expectativas de desempenho do usuário? E a lista continua. Sem um entendimento claro de quais desafios são esses e o que é preciso fazer para endereçá-los prontamente em seu ciclo de desenvolvimento, tentar a retro compatibilidade de soluções tardias pode tornar-se caro e complicado. O *Guia de Arquitetura e Projeto de Smart Clients* o ajudará a descobrir os desafios de projeto e o guiará em direção às soluções corretas para o seu projeto. Este é precisamente o tipo de informação que os consumidores têm pedido e estou muito ansioso para ver este guia publicado.

Divirta-se construindo aplicações clientes novamente!

## **Mark Boulter**

### **Coordenador Técnico de Gerenciamento de Programa**

***Mark Boulter** é Gerente de Programa Sênior na equipe .NET Client da Microsoft. Ele trabalhou no Windows Forms e outras bibliotecas de classes relacionadas desde que entrou na Microsoft. Antes de unir-se à Microsoft, Mark trabalhou como consultor para a ParcPlace Systems, no Reino Unido, ajudando consumidores a construírem sistemas de servidores de clientes e de análise de dados em Smalltalk. Antes disso, Mark passou muitos anos, mais do que se permite admitir, na IBM do Reino Unido, trabalhando em uma grande variedade de projetos, incluindo sistemas de servidor de clientes de larga escala, ferramenta CASE, um mecanismo de fluxo de trabalho e sistemas de gerenciamento de pedido. Os interesses de Mark incluem escutar música New Wave Industrial Pós-punk e Blues, ler qualquer coisa e cuidar de seus gatos.*

# 1

## Introdução

Bem-vindo ao Guia de Arquitetura e Projeto de Smart Clients. As Aplicações Smart Client são uma alternativa poderosa a aplicações Thin Client (Aplicações Web). Elas podem dar ao usuário uma rica e receptiva interface com o usuário, a possibilidade de trabalhar offline e uma forma de aproveitar melhor os recursos locais de hardware e software. Além disso, podem ser projetadas para executar em um largo espectro de dispositivos de cliente, incluindo PCs Desktop, Tablets e pequenos dispositivos móveis como Pocket PCs e Smartphones.

Os Smart Clients oferecem aos usuários acesso a informações e serviços remotos dentro de um ambiente de cliente poderoso e intuitivo, e são uma solução efetiva para aplicações flexíveis direcionadas ao usuário e para o aumento de sua produtividade e satisfação.

Aplicações Smart Client podem ser projetadas para combinar os benefícios tradicionais de uma aplicação Rich Client com os benefícios de gerenciamento de uma aplicação Thin Client. Entretanto, para compreender totalmente os benefícios de uma aplicação Smart Client, é necessário considerar um grande número de questões de arquitetura e projeto. Este guia demonstra os desafios de arquitetura e design encontrados ao projetar e implementar uma aplicação Smart Client, e mostrará como superar esses desafios, permitindo que sejam percebidos os benefícios de uma aplicação Smart Client no menor espaço de tempo possível.

---

**Observação:** Recursos técnicos adicionais para Smart Client estão disponíveis no Smart Client Developer Center em: <http://msdn.microsoft.com/smartclient/>. O valor de negócios do Smart Client é discutido no site .NET da Microsoft em <http://www.microsoft.com/net/smartclient/default.mspx>.

---

## O que é um Smart Client?

Para compreender totalmente como o Smart Client combina os benefícios de Rich Client e Thin Client, é necessário examinar a história e princípios subjacentes por trás dos modelos de aplicações Rich e Thin Client e revisar algumas vantagens e desvantagens associadas a cada um deles.

### Aplicações Rich Client

Em meados dos anos 90, o número de aplicações Rich Client desenvolvidos para o sistema operacional Microsoft® Windows® aumentou drasticamente. Esses clientes eram criados para tirar vantagem dos recursos locais de hardware e dos recursos da plataforma de sistema operacional do cliente.

Apesar da funcionalidade impressionante de muitas dessas aplicações, elas têm suas limitações. Muitas são independentes e operam no computador cliente com pouco ou nenhum conhecimento do ambiente no qual operam. Esse ambiente inclui os demais computadores e qualquer serviço na rede, bem como qualquer outra aplicação no computador do usuário. Frequentemente, a integração entre aplicações é limitada ao uso dos recursos de recortar ou copiar e colar oferecidas pelo Windows para transferir pequenas quantidades de dados entre as aplicações. Há tecnologias para o aumento de conectividade de aplicações Rich Client. Por exemplo, aplicações de duas camadas permitem que diversos usuários acessem dados comuns da rede, e o DCOM permite às aplicações tornarem-se mais distribuídas (com o DCOM, a lógica e o estado não estão mais atreladas ao computador cliente, e sim encapsuladas dentro de objetos distribuídos por diversos computadores).

Entretanto, aplicações conectadas são consideravelmente mais complexas de se desenvolver. Conforme cresce o tamanho e a complexidade dessas aplicações distribuídas, qualquer conexão entre aplicações cliente e de serviços utilizados torna-se gradativamente mais difícil de manter. Enquanto Rich Clients fornece geralmente uma experiência de usuário de alta qualidade e com boa resposta e possui bom suporte a desenvolvedores e plataformas; eles também são difíceis de implementar e manter. Conforme a complexidade das aplicações e plataformas de clientes aumenta, também aumenta as dificuldades associadas à implementação da aplicação no computador cliente de maneira segura e confiável. Uma aplicação pode facilmente travar outra se um componente ou biblioteca compartilhada incompatível comum for implementada, fenômeno conhecido como fragilidade de aplicação. Novas versões da aplicação são geralmente disponibilizadas através da nova implementação de toda a aplicação, o que pode aumentar um problema de fragilidade da aplicação.

## **Aplicações Thin Client**

A internet oferece uma alternativa ao modelo tradicional de Rich Client que resolve muitos dos problemas associados à implementação e manutenção da aplicação. Aplicações Thin Client baseadas em navegador são aplicadas e atualizadas em um servidor da Web central; portanto, eles excluem a necessidade de implementar e gerenciar explicitamente qualquer parte da aplicação no computador cliente.

Este modelo permite que as empresas verifiquem eficientemente a exposição de suas aplicações a uma audiência externa grande e variada. Como os Thin Clients provaram ser eficientes na solução de alguns dos problemas de aplicação e gerenciamento, são utilizados atualmente para fornecer acesso a muitas aplicações de linha de negócios (LOB Line of Business) e para usuários dentro de uma organização, assim como acesso a aplicações externas, para consumidores e parceiros. Tudo isso, apesar do fato de que as necessidades e expectativas desses dois tipos de usuários são, em geral, radicalmente distintas.

Aplicações Thin Client têm algumas desvantagens. O navegador deve ter conexão de rede o tempo todo. Isso significa que usuários móveis não têm acesso a aplicações caso estejam desconectados, tendo que inserir dados novamente ao retornar ao escritório. Outro fato é que recursos comuns da aplicação como “arrastar e soltar” (drag-and-drop) podem não estar disponíveis, o que pode reduzir o aproveitamento da aplicação.

Devido à vasta maioria de lógica de aplicação no servidor, Thin Clients fazem solicitações frequentes ao servidor para solicitação de dados e processamento. O navegador deve esperar por uma resposta antes que o usuário possa continuar a utilizar a aplicação; portanto, a aplicação geralmente terá uma resposta pior do que uma aplicação Rich Client equivalente. Esse problema é mais evidente em condições de banda estreita ou alta latência, e os problemas de desempenho resultantes podem levar a uma redução significativa no aproveitamento da aplicação e eficiência do usuário. Uma aplicação LOB que necessite de entrada carregada de dados e/ou frequente navegação por muitas janelas pode ser particularmente afetada por esse problema.

## Aplicações Smart Client

As aplicações Smart Client podem ser criadas para combinar os benefícios de aplicações Rich Client com os pontos fortes da implementação e gerenciabilidade de uma aplicação Thin Client, embora a natureza precisa do equilíbrio entre as duas abordagens dependa do cenário correto.

Aplicações Smart Client freqüentemente têm requerimentos distintos e portanto, variam bastante em projeto e implementação. Entretanto, todos os Smart Clients compartilham algumas (ou todas) as características abaixo:

- Utilizam recursos locais;
- Utilizam recursos da rede;
- Suportam usuários conectados ocasionalmente;
- Fornecem instalação e atualização inteligentes;
- Oferecem flexibilidade de dispositivos clientes.

Muitas aplicações não necessitam de todas essas características. Conforme seu Smart Client seja projetado, será preciso considerar cuidadosamente o cenário de sua aplicação e decidir quais dessas características sua aplicação Smart Client necessita. Incorporar todas essas características em sua aplicação demandará um planejamento e projeto cuidadoso e, em muitos casos, serão necessários recursos de implementação significativos.

---

**Observação:** O .NET Framework ajuda você a implementar muitas das características de uma aplicação Smart Client. Montagens coesas, junto com a sustentação de instalações isoladas, lado a lado e de versões múltiplas de uma aplicação, ajudam a reduzir os problemas de distribuição e de fragilidade da aplicação associados aos Rich Clients. A biblioteca de base do .NET Framework fornece a sustentação extensiva para a interação com os Web Services, e fornece Windows Form. Usando o tempo de execução da língua comum (CLR Common Language Runtime), você pode utilizar qualquer linguagem suportada pelo .NET para desenvolver suas aplicações Smart Client.

---

## Utilizando Recursos Locais

Uma aplicação Smart Client bem projetada tira máxima vantagem do fato de que códigos e dados são implementados no cliente e executados e acessados localmente. Ela fornece uma aplicação com uma interface de usuário rica e responsiva, e uma poderosa capacidade de processamento no cliente. Pode, por exemplo, possibilitar que o usuário execute manipulações complexas de dados, visualização, busca ou operações de organização.

Os Smart Clients podem tirar vantagem dos recursos de hardware do cliente (como telefones ou leitores de código de barras) e de outros softwares e aplicações. Isso os torna aptos a resolver problemas que uma aplicação Thin Client não conseguiria resolver tão bem, como aplicações de pontos de venda. Os Smart Clients também podem se beneficiar dos softwares locais, como os aplicativos do Microsoft Office, ou qualquer aplicação LOB instalada no computador cliente. Criar soluções que integram e coordenam múltiplas aplicações LOB permite aos usuários trabalhar com maior eficiência, tomar melhores decisões e reduzir erros de entrada de dados. Essas soluções também podem permitir que sua aplicação se integre melhor ao ambiente de trabalho do usuário — por exemplo, por meio de uma interface padrão ou familiar — o que pode levar a uma diminuição nos custos de treinamento.

Outras aplicações clientes podem ser integradas ou coordenadas por um Smart Client para fornecer soluções gerais coerentes e eficientes. Essas aplicações também deveriam conhecer o contexto na qual estão sendo empregadas e adaptar-se a esse contexto para ajudar o usuário sempre que possível, por exemplo, armazenando preemptivamente dados úteis em cachê, de acordo com o modelo de uso na função do usuário.

Maximizar o uso e integrar recursos locais às aplicações Smart Client permite que estas façam uma melhor e mais eficiente utilização do hardware que já está disponível. Frequentemente, poder de processamento, memória e capacidades gráficas avançadas acaba não sendo utilizado. Utilizar os recursos no computador cliente também pode reduzir os requisitos de hardware no servidor.

## **Utilizando Recursos de Rede**

Os Smart Clients podem consumir e utilizar diferentes serviços e dados na rede. Eles são uma maneira eficiente de recuperar dados de muitas fontes distintas e podem ser projetados para analisar ou agregar dados, permitindo que o usuário tome decisões mais eficientes e melhor informadas. Por exemplo, um Smart Client poderia usar um serviço de mapeamento para fornecer detalhes sobre localização e orientações de endereço. Aplicações Smart Client devem estar conectadas o máximo de tempo possível e utilizar os recursos e serviços disponíveis na rede. Elas não devem ser aplicações independentes e deveriam sempre ser parte de uma solução distribuída mais abrangente. No mínimo, uma aplicação Smart Client deveria utilizar serviços centralizados que ajudem a manter a implementação e fornecer serviços de aplicação e atualização.

A natureza conectada de uma aplicação Smart Client permite que ela forneça uma valiosa concentração de dados, análise e serviços de transformação. Esta concentração pode permitir que os usuários colaborem em tarefas em tempo real durante um determinado período. Em muitos casos, uma aplicação Smart Client pode fornecer recursos de portal ao usuário, permitindo que diversos dados e serviços sejam coordenados e integrados em uma solução completa.

Para maiores detalhes sobre como projetar seu Smart Client para fazer uso de serviços conectados, consulte: "Capítulo 3, Conectando-se."

## **Oferecendo Suporte aos Usuários Ocasionalmente Conectados**

Os Smart Clients podem ser projetados para fornecer funcionalidade aos usuários que estejam ocasionalmente conectados à rede, permitindo que o usuário continue a trabalhar eficientemente quando offline, em condições de banda estreita ou rede de alta latência, ou quando a conectividade for intermitente. Para aplicações móveis, Smart Clients também podem otimizar a largura de banda, por exemplo, enviando blocos de solicitações ao servidor para fazer um melhor uso da conexão.

Mesmo quando o cliente está conectado à rede durante a maior parte do tempo, as aplicações Smart Client podem melhorar o desempenho e o aproveitamento criando cachê de dados e gerenciando a conexão de forma inteligente. Em um ambiente de banda estreita ou alta latência, por exemplo, uma aplicação Smart Client pode gerenciar a conexão de tal forma que o aproveitamento e a responsividade da aplicação não são prejudicados e o usuário pode continuar a trabalhar eficazmente.

A possibilidade de se trabalhar desconectado ou ocasionalmente conectado aumenta a produtividade e a satisfação do usuário. Uma aplicação Smart Client deveria buscar fornecer o máximo de funcionalidade possível quando utilizada offline.

Para obter detalhes adicionais sobre como projetar sua aplicação Smart Client para oferecer suporte a usuários conectados ocasionalmente, consulte: Capítulo 4, "Smart Clients Conectados Ocasionalmente."

## **Fornecendo Instalação e Atualização Inteligentes**

Alguns dos maiores problemas com os Rich Clients tradicionais ocorrem quando a aplicação é implementada ou atualizada. Muitas aplicações Rich Client têm um grande número de complexos requerimentos de instalação e pode compartilhar códigos registrando componentes e/ou instalando DLLs em um local comum, resultando em fragilidade da aplicação e dificuldades de atualização.

Aplicações Smart Client podem ser projetadas para gerenciar sua própria implementação e atualização de maneira muito mais flexível e inteligente que aplicações Rich Client tradicionais. Elas podem evitar esses problemas comuns, o que pode ajudar a reduzir os custos de gerenciamento de sua aplicação.

Há muitas maneiras de se implementar Smart Clients. Isso inclui a simples cópia de arquivos em um computador local, o download automático de códigos a partir de um servidor central utilizando uma implementação no-touch ou implementando-se o pacote de instalação do Windows utilizando tecnologia de ponta para empresas, como o Microsoft Systems Management Server (SMS). O método escolhido dependerá de sua situação específica.

Aplicações Smart Client podem atualizar-se automaticamente, seja durante sua execução ou em segundo plano. Esse recurso permite que elas sejam atualizadas em uma base de função por função; atualizado de forma encenado, permitindo que as aplicações sejam enviadas para grupos piloto ou um conjunto limitado de usuários; ou atualizado de acordo com um planejamento estabelecido.

O .NET Framework permite que se nomeiem os componentes da aplicação, o que significa que ele poderá especificar e executar as versões exatas dos componentes com os quais foi construído e testado. O .NET Framework permite que aplicações sejam isoladas para que a instalação de uma aplicação não prejudique a de outra, e para que diferentes versões da mesma aplicação possam ser implementadas lado a lado. Esses recursos simplificam bastante a implementação da aplicação e removem muitos dos problemas de fragilidade da aplicação associados às aplicações Rich Client.

Para obter mais informações sobre instalação inteligente e atualizações, consulte Capítulo 7, "Implementação."

## Fornecendo Flexibilidade para Dispositivos de Cliente

Os Smart Clients podem oferecer um ambiente de cliente flexível e personalizável, permitindo que o usuário configure a aplicação para oferecer suporte a seu modo de trabalho preferido. Aplicações Smart Client não se restringem a desktops ou laptops. Com o aumento da capacidade de conectividade dos dispositivos de menor escala, a necessidade de aplicações cliente úteis e que forneçam acesso a dados e serviços essenciais em dispositivos diversos também aumenta. Ao lado do .NET Compact Framework, o .NET Framework oferece uma plataforma comum onde aplicações Smart Client podem ser criadas.

Smart Clients podem ser projetados para adaptar-se ao ambiente do host local, oferecendo funcionalidade apropriada ao dispositivo no qual são executados. Por exemplo, uma aplicação Smart Client projetada para ser executada em um Pocket PC deveria fornecer uma interface de cliente adequada para utilizar uma caneta Stylus em uma pequena área da tela.

Em muitos casos, será preciso projetar diversas versões de uma aplicação Smart Client, cada uma visando um tipo específico de dispositivo para tirar vantagem máxima dos recursos particulares para os quais cada dispositivo possui suporte. Como os dispositivos de menor escala são tipicamente limitados em sua habilidade de entregar um grande número de recursos das aplicações Smart Client, eles podem fornecer acesso móvel a apenas um subconjunto de dados e serviços que somente uma aplicação Smart Client oferece ou ser utilizados para coletar e agregar dados quando o usuário estiver móvel. Esses dados podem ser analisados ou processados por uma aplicação Smart Client com mais recursos ou por uma aplicação de servidor.

Uma percepção no ambiente dos recursos e utilização de um determinado dispositivo de destino, seja um desktop, laptop, tablet ou dispositivo móvel, e a habilidade de projetar a aplicação de modo a fornecer funcionalidade apropriada são recursos essenciais de muitas aplicações Smart Client.

---

**Observação:** Este guia não abrange detalhes específicos de design e arquitetura para o desenvolvimento de aplicações Smart Client para execução em dispositivos móveis, porém, muitos dos tópicos abordados são igualmente relevantes, mesmo que a aplicação seja executada em um desktop ou em outro dispositivo.

---

## Tipos de Smart Clients

Smart Clients variam imensamente em design e implementação, tanto em requerimentos de aplicação quanto em número de cenários e ambientes nos quais podem ser usados. Smart Clients podem, portanto, adotar diferentes formas e estilos. Essas formas podem ser divididas em três categorias mais amplas, de acordo com a plataforma de destino da aplicação Smart Client:

- Aplicações Smart Client do Windows;
- Aplicações Smart Client do Office;
- Aplicações Smart Client Móveis.

É comum para uma aplicação Smart Client ter como destino uma ou mais dessas plataformas, dependendo da função do usuário e a funcionalidade necessária. Essa flexibilidade é um dos pontos fortes das aplicações Smart Client.

O restante deste guia se concentra em assuntos comuns a todos os três tipos de aplicações Smart Client, em vez de oferecer explicações detalhadas de problemas que afetam cada categoria individualmente. Entretanto, é útil examinar brevemente cada tipo para que seja possível determinar qual estilo de aplicação será melhor para a sua situação.

### Aplicações Smart Client do Windows

Quando você pensa em uma aplicação Rich Client, deve geralmente pensar em uma aplicação de desktop que utiliza os recursos de sistema disponíveis e que oferece uma interface de usuário rica. Aplicações Smart Client para Windows representam uma evolução das aplicações Rich Client tradicionais e fornecem funcionalidades específicas.

Esses tipos de aplicações geralmente utilizam o Windows Forms para oferecer uma interface de usuário familiar no estilo Windows, na qual a própria aplicação fornece grande parte da funcionalidade e não depende de outra para oferecer a interface de usuário principal. Esses Smart Clients podem ir desde aplicações simples implementados em HTTP até aplicações muito sofisticadas.

Uma aplicação Smart Client do Windows é adequada em situações no qual a aplicação necessita ser implementada e acessada como uma aplicação familiar do tipo desktop. Esses tipos fornecem, geralmente, a maioria de suas funcionalidades sozinhas e podem integrar ou coordenar outras aplicações quando apropriado. Elas proporcionam funcionalidade de aplicação em sintonia com tarefas particulares para oferecer processamento específico ou de alto desempenho ou capacidades gráficas. Aplicações Smart Client do Windows são, em geral, mais adequadas para aplicações executadas em Desktops, laptops ou tablet PCs. Além disso, elas oferecem funcionalidades que não estão associadas com um documento ou tipo de documento em particular.

Esses tipos de aplicações Smart Client do Windows podem ser utilizadas em uma grande variedade de situações, por exemplo, como uma aplicação LOB, financeira, científica ou colaborativa. Exemplos desses tipos de aplicação são o Microsoft Money e o Microsoft Outlook® Messaging and Collaboration Client.

### Aplicações Smart Client do Office

O Microsoft Office System 2003 oferece uma plataforma eficaz para construção de aplicações Smart Client, especialmente em ambiente corporativo. Com uma solução de Smart Client do Office, é possível integrar origens de dados, acessadas por meio de serviços da Web, com os recursos do Word 2003, Excel 2003, InfoPath 2003 ou qualquer um das aplicações do Office para desenvolver soluções de Smart Clients.



Essas aplicações Smart Client do Office podem tornar-se parte integrada de um ciclo de gerenciamento de informações de uma organização e não somente um local estático para dados de documentos. Elas podem fornecer dados sensíveis a contexto conforme o usuário trabalhe em um documento, bem como fluxo de trabalho e direcionamento de tarefas, análise de dados, colaboração, relatório e recursos de apresentação que transformam dados exibidos pelos Web Services em informações úteis.

O Microsoft Office suporta XML e separa os dados de outros aspectos de um documento para que eles possam ser reutilizados por outras aplicações. Como os dados da aplicação no Microsoft Office podem ser descritos pelo mesmo esquema XML definido pelo cliente através de aplicações múltiplas, os desenvolvedores podem integrar esses dados a aplicações Smart Client.

O Microsoft Office 2003 tem muitos recursos chave e opções para a construção de soluções Smart Client, incluindo:

- **Smart tags.** Smart tags dão à aplicação uma maneira de fornecer o usuário com dados sensíveis a contexto relativo ao conteúdo de um documento e permite fácil visualização e utiliza informações relevantes quando trabalha dentro de um documento. Por exemplo, smart tags podem ser utilizadas para oferecer posição de conta para os consumidores conforme eles são mencionados dentro de um documento ou podem ser utilizados para oferecer informação de posição de pedido conforme a identificação de um pedido é digitada. Esse feedback contextualizado permite que o usuário tome decisões mais bem informadas, conforme trabalha.
- **Smart documents.** Smart documents oferecem uma maneira mais poderosa para que o usuário interaja com Web Services e documentos de negócios. Os Smart documents são um novo tipo de modelo de solução para Word 2003 e Excel 2003, têm uma estrutura XML subjacente e um painel de tarefas que pode ser utilizado para exibir informações de contexto, tarefas, ferramentas, próximos passos e outras informações relevantes para o usuário. O usuário poderá iniciar outras tarefas e ações interagindo com o painel de tarefas, permitindo que soluções de negócio abrangentes sejam construídas.
- **Ferramentas do Microsoft Visual Studio® para o Microsoft Office System.** Esse conjunto de ferramentas permite que os desenvolvedores criem aplicações Smart Client do Office de código gerenciado utilizando o sistema de desenvolvimento Microsoft Visual Studio .NET 2003. Os desenvolvedores podem separar soluções de documento a partir dos códigos subjacentes (uma alternativa a modelos anteriores de Smart Client que continham macros em Visual Basic for Applications com lógica personalizada). Utilizar códigos gerenciados, o Microsoft Office fornece aos desenvolvedores opções mais efetivas para criar, implementar e gerenciar atualizações para soluções Smart Client.
- **Microsoft Office InfoPath™ 2003.** O InfoPath 2003 é uma aplicação que pode reunir dados estruturados do usuário utilizando uma interface de formulário. Ele suporta Web Services XML, uma interface de usuário com base em formulários e oferece suporte para tecnologias padrão como WSDL e UDDI. O InfoPath 2003 também suporta uso offline limitado, permitindo que o usuário interaja com o formulário quando está offline para depois permitir que ele envie o formulário a um serviço da Web quando estiver on-line.

Este guia não pretende abordar qualquer um dos assuntos específicos sobre Smart Clients do Office, mas a maioria dos tópicos cobertos é completamente relevante às aplicações Smart Client discutidas acima.



## Aplicações Móveis Smart Client

Os Smart Clients móveis são aplicações que são executadas em dispositivos inteligentes, Pocket PCs, Smartphones e outros pequenos dispositivos. Essas aplicações são desenvolvidas utilizando o .NET Compact Framework, que é um subconjunto do .NET Framework completo. O .NET Compact Framework contém muitos dos atributos do .NET Framework completo, suporta XML e utiliza serviços da Web. É otimizado para uso em pequenos dispositivos e inclui o Windows Forms Designer para desenvolvimento da interface de usuário.

Utilizando o Visual Studio .NET Smart Device Projects, pode-se desenvolver Smart Clients que serão executados no .NET Compact Framework. Essa abordagem permite desenvolver, testar e depurar uma aplicação utilizando o Visual Studio .NET em um emulador de pequenos dispositivos. O uso de um emulador aumenta significativamente a velocidade de desenvolvimento e testes desses tipos de aplicações.

Aplicações Smart Client móveis são utilizadas geralmente para oferecer acesso móvel a dados e servidores essenciais ou para coletar e agregar dados quando o usuário está em movimento. Exemplos desses tipos de aplicações são: aplicações que coletam e agregam dados financeiros e de seguro, aplicações de gerenciamento de estoque e aplicações de gerenciamento de produtividade pessoal.

Este guia não se concentra em aplicações Smart Client móveis, embora muitos dos assuntos de arquitetura e soluções discutidas sejam relevantes para dispositivos inteligentes.

## Escolhendo entre Smart Clients e Thin Clients

Para a escolha da arquitetura de aplicação correta para sua situação, deve-se considerar diversos fatores. Para determinar se uma abordagem Smart Client é a mais apropriada para sua aplicação, considere cuidadosamente as necessidades atuais e futuras de sua aplicação de negócios. Se sua aplicação for baseada em uma arquitetura inadequada, poderá não atender aos requisitos e expectativas dos usuários e do negócio como um todo. Mudar a arquitetura posteriormente para satisfazer novos requisitos ou tirar vantagens de novas oportunidades pode ser extremamente caro.

Uma arquitetura Thin Client é, freqüentemente, a mais apropriada se você precisa tornar uma aplicação externa disponível para um grupo externo diverso, enquanto uma arquitetura Smart Client é geralmente a mais apropriada para uma aplicação interna que necessita integrar-se com ou coordenar outras aplicações ou hardware do cliente, ou que seja requerido que trabalhe offline ou forneça funcionalidade de alto desempenho específico por meio de uma interface de usuário com boa resposta.

Na realidade, essas duas abordagens se sobrepõem em grande parte e cada uma tem suas vantagens e desvantagens. Você só conseguirá escolher a abordagem correta depois de considerar cuidadosamente suas necessidades e entender como cada abordagem se aplica à sua situação. Use a Tabela 1.1 como auxílio para escolher entre a arquitetura Smart Client e a Thin Client.

Tabela 1.1: Recursos de Thin Clients e Smart Clients

Feature	Thin client	Smart client
Fornecer uma rica interface de usuário	Sim, mas difícil de desenvolver, testar e depurar. Geralmente atrela a aplicação a um único navegador	Sim. Mais fácil de desenvolver, testar e depurar
Pode tirar vantagem dos recursos de hardware no computador local	Sim, mas somente através de componentes COM	Sim

(continua)

Feature	Thin client	Smart client
Pode interagir com outras aplicações locais	Não	Sim
Pode ser multi-threaded	Não	Sim
Funciona offline	Não	Sim
Tem bom desempenho em ambientes de banda estreita e de alta latência	Não	Sim
Fácil de implementar	Sim	Variações. Dificuldade depende dos requisitos da aplicação
Baixos custos de manutenção e mudança de gerenciamento	Sim	Variações. Custos dependem das Solicitações da aplicação
Pode ser implementado para uma larga variedade de clientes com várias habilidades	Sim, apesar de mais complexos Thin Clients podem necessitar um navegador único.	Sim. Pode ser implementado em qualquer plataforma que suporte .NET Framework (incluindo o .NET Compact Framework).

## Desafios da Arquitetura Smart Client

Os desafios da arquitetura de um Smart Client diferem dos desafios de um Thin Client, e você precisará se preparar para eles durante o projeto da aplicação. Os benefícios de aplicações Smart Client são significativos, mas você os entenderá somente se cuidar desses desafios da maneira correta.

Smart Clients permitem que dados e lógica sejam distribuídos para o computador cliente, enquanto Thin Clients mantêm dados e lógica centralizados no servidor Web e em outros serviços back-end. Embora a abordagem Smart Client permita que você torne a aplicação mais eficiente, sem voltas ao servidor para determinar os passos seguintes, deve-se levar em conta que a aplicação e seus dados estão mais amplamente distribuídos do que em aplicações Thin Client e modificar seu projeto de acordo. Se estiver implementando regras de negócios no cliente, será necessário atualizá-las conforme solicitado, sem atualizar toda a aplicação. Isso significa ter que utilizar mecanismos diferenciadores para atualizar a aplicação e atualizar as regras de negócios dentro da aplicação.

Colocar dados em cachê no cliente, fará com que o desempenho e o aproveitamento melhorem significativamente, mas você deve assegurar que os dados sejam atualizados adequadamente e os que estejam desatualizados não sejam utilizados. Como muitos usuários podem acessar e utilizar os mesmo dados, você também deve considerar os efeitos de dados simultâneos. Sua aplicação deve estar capacitada para lidar com conflito de dados ou problemas de reconciliações que eventualmente possam surgir, já que a aplicação agora é amplamente distribuída e pode operar offline. O Capítulo 3, "Manipulando Dados," trata desse assunto mais profundamente.

O .NET Framework oferece uma grande flexibilidade na maneira como suas aplicações Smart Client serão hospedadas. As aplicações podem ser executadas como aplicações desktop tradicionais ou podem ser hospedadas dentro do Office ou do Microsoft Internet Explorer. Muitas combinações são possíveis. Por exemplo, uma aplicação Windows Forms pode hospedar componentes do Internet Explorer ou do Office, e qualquer host pode incluir qualquer outra aplicação. Você pode seguir partes lógicas voláteis da aplicação (como regras de negócio controlando descontos para pedidos de grande volume) em montagens que são transferidas quando necessário através do HTTP.

Isso torna óbvia a necessidade de implementar novas versões da aplicação cliente conforme novas lógicas de aplicação são desenvolvidas. Pode utilizar o mesmo modelo para atributos de aplicação adicionais (ou raramente utilizados), para que o tamanho inicial da aplicação seja mantido ao mínimo e recursos adicionais sejam instalados de acordo com a necessidade.

Pode-se decidir implementar seu Smart client como uma aplicação combinada, em que muitas aplicações se combinam para formar uma solução coerente. Tais soluções podem ser formadas agrupando-se aplicações de desktop ou fornecendo uma aplicação de formato genérico que inclua múltiplas aplicações leves que juntas formam uma solução.

As aplicações combinadas são especialmente úteis em situações nas quais os usuários têm que acessar muitas aplicações para realizar seu trabalho. Por exemplo, agentes de atendimento ao consumidor em call centers geralmente trabalham com muitas aplicações LOB, incluindo desktop, baseado em navegador e aplicações com base no terminal. Todas essas aplicações podem ser hospedadas dentro de uma aplicação genérica do Windows Forms, que oferece integração entre eles, simplificando enormemente o trabalho do usuário e, mais importante, reduzindo o tempo gasto em uma chamada em particular. Ao oferecer um formato genérico para hospedar essas aplicações LOB, recursos comuns de infra-estrutura, como segurança, aplicação, gerenciamento de janelas, integração de aplicações, auditoria e assim sucessivamente, podem ser desenvolvidas, testadas e reutilizadas em distintas soluções, permitindo que os desenvolvedores de aplicações LOB concentrem-se na funcionalidade do negócio.

O advento de arquiteturas orientadas a serviços significa que você pode projetar Smart Clients para utilizar serviços de rede. Todos esses serviços são fornecidos em padrões da indústria, o que melhora a interoperabilidade, suporte à ferramenta de programação e a facilidade com que novos atributos podem ser desenvolvidos em uma aplicação Smart Client.

## Escopo deste Guia

Este guia concentra nos assuntos de arquitetura e projeto das aplicações Smart Client construídos com as tecnologias Microsoft .NET. Ele supõe que se esteja construindo as aplicações Smart Client utilizando o Microsoft .NET Framework e utilizando o Microsoft .NET Windows Forms para construção de qualquer interface de usuário.

Este guia não trata profundamente de assuntos relacionados à implementação. Os detalhes sobre a implementação de uma aplicação Smart Client no Microsoft Office 2003 ou em um dispositivo móvel, embora muitos dos assuntos abordados neste guia sejam relevantes à aplicações Smart Client.—  
Sejam elas Windows Forms independentes, Office, ou de dispositivos móveis.

## Como utilizar este Guia

Este guia foi desenvolvido para ser utilizado de duas formas. Primeiramente, o guia é estruturado para oferecer uma visão geral razoavelmente abrangente dos assuntos de arquitetura e design com os quais você poderá deparar-se durante a construção de uma aplicação Smart Client. A leitura deste guia do começo ao fim dará um entendimento completo dos problemas com os quais você poderá deparar-se e como superá-los.

Como alternativa, caso prefira explorar profundamente os assuntos acerca de um tópico específico, pode-se ler os capítulos individualmente e analisar independentemente os temas relevantes.

## Quem Deve Ler Este Guia

Este guia destina-se a arquitetos de software e desenvolvedores que estejam construindo aplicações Smart Client em tecnologias Microsoft .NET.

### Pré-requisitos

Para se beneficiar completamente deste guia, você já deve possuir uma compreensão das seguintes tecnologias e conceitos:

- Microsoft .NET Framework
- Ferramenta de desenvolvimento Microsoft Visual Studio .NET 2003
- Ferramenta de desenvolvimento Microsoft® Visual C#®
- Extensible Markup Language (XML)
- Message Queuing (MSMQ)
- Multithreading
- Operação de Banco de Dados Relacional
- Arquitetura e Projeto de Aplicações Distribuídas

---

**Observação:** Para mais informações a respeito de Arquitetura e Projeto de Aplicações Distribuídas, consulte <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxoriDesignConsiderationsForDistributedApplications.asp> e <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxoriplanningdistributedapplications.asp>.

---

## Resumo de Capítulos

Este guia é composto pelos capítulos abaixo; cada um dos quais abordando um tema específico relevante para Smart Clients. Cada capítulo foi escrito para ser lido completo ou em partes, de acordo com sua necessidade.

### Capítulo 1: Introdução

Este capítulo oferece uma descrição de alto nível sobre as aplicações Smart Client e descreve algumas de suas propriedades básicas e benefícios. Ele também discute alguns dos problemas de arquitetura de alto nível e oferece uma orientação para ajudá-lo a determinar se uma arquitetura Smart Client é ideal para sua aplicação.

### Capítulo 2: Manipulação de Dados

Em Smart Clients, os dados da aplicação estão disponíveis no cliente. Esses dados precisam ser gerenciados corretamente para assegurar que eles permaneçam válidos, consistentes e seguros. Se os dados forem fornecidos por uma aplicação de servidor, a aplicação Smart Client pode colocar em cache os dados para melhorar o desempenho ou permitir o uso quando offline. Se sua aplicação Smart Client oferecer a possibilidade de modificar dados localmente, as mudanças no cliente têm que ser sincronizadas com a aplicação do servidor posteriormente. Este capítulo examina as diversas considerações para a manipulação de dados no cliente, incluindo chave de dados, simultaneidade de dados e o uso de datasets e vínculo de dados no Windows Forms.

### Capítulo 3: Conectando-se

Aplicações Smart Client geralmente formam parte de uma aplicação distribuída maior, portanto, freqüentemente estão conectados a uma rede e interagem com recursos desta, como Serviços da Web, juntamente com componentes ou processos no próprio computador cliente. Este capítulo descreve as várias formas de conectar sua aplicação e utilizar esses recursos, além de discutir os pontos fortes e fracos de cada uma dessas formas.

### Capítulo 4: Smart Clients Conectados Ocasionalmente

Este capítulo traz uma discussão sobre os problemas enfrentados durante o projeto e a construção de aplicações Smart Client conectadas ocasionalmente à rede. Aborda o conceito de conectividade, descreve as duas abordagens principais para implementação de recursos Offline e discute algumas dos pontos que são necessários considerar para tornar sua aplicação disponível offline.

## Capítulo 5: Considerações de Segurança

Este Capítulo abrange temas de segurança de Smart Client. Os Smart Clients distribuem dados e lógica para o computador cliente, portanto, as preocupações com segurança são diferentes das associadas a uma aplicação Thin Client, na qual os dados e lógica ficam mais confinados no servidor. Este capítulo discute segurança de dados, autenticação, autorização e a função da segurança de acesso do código em uma aplicação Smart Client.

## Capítulo 6: Utilizando Threads Múltiplos

Este capítulo discute os assuntos que envolvem o uso de threads múltiplos em aplicações Smart Client. Para maximizar a responsividade de suas aplicações Smart Client, é necessário considerar cuidadosamente como e quando utilizar threads múltiplos. Threads podem melhorar significativamente o aproveitamento e desempenho de sua aplicação, porém requerem cuidadosa análise para determinar como irão interagir com a interface de usuário.

## Capítulo 7: Implementando e Atualizando Aplicações Smart Client

Smart Clients não sofrem dos tradicionais problemas de implementação e atualização associados às aplicações Rich Client. Os recursos fornecidos pelo .NET Framework e pela plataforma Windows ajudam a evitar muitos problemas associados a aplicações Rich Client tradicionais. Este capítulo descreve como melhor utilizar esses atributos e como escolher entre os mecanismos de aplicação e atualização disponíveis.

## Capítulo 8: Performance da Aplicação Smart Client

Este capítulo examina técnicas que podem ser utilizadas ao arquitetar e projetar sua aplicação Smart Client para assegurar que seu desempenho seja otimizado. Esta seção demonstra as ferramentas e técnicas que podem ser utilizadas para identificar problemas de desempenho em suas aplicações Smart Client.

## Sumário

Tanto Thin Clients como Smart Clients podem ser utilizados para oferecer aplicações LOB para sua empresa. Entretanto, cada tipo de cliente tem suas vantagens e desvantagens. Durante o projeto de sua aplicação, é importante considerar cuidadosamente as especificidades de sua situação antes de determinar qual é a mais apropriada. Este capítulo explicou como Smart Clients se desenvolvem, bem como os seus recursos associados. Agora, você pode utilizar o restante deste guia para determinar como projetar e implementar Smart Clients em sua empresa.

## Informações Adicionais

Os recursos abaixo oferecem mais informações sobre práticas & padrões, Smart Clients e outros blocos de aplicações que podem ser utilizados para orientações específicas.

- Web site práticas & padrões em <http://www.microsoft.com/resources/practice/default.mspix>
- Biblioteca Práticas & Padrões em <http://www.microsoft.com/resources/practices/completelist.asp>
- Visão Geral das aplicações Smart Client no Microsoft Office System no MSDN® em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc\\_ip2003\\_ta/html/odc\\_IPOffice2003SmartClient.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc_ip2003_ta/html/odc_IPOffice2003SmartClient.asp)
- Arquitetura de Aplicação para.NET: Projetando Aplicações e Serviços em MSDN em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp>

# 2

## Manipulação de Dados

Em Smart Clients, os dados da aplicação estão disponíveis no cliente. Para seus Smart Clients funcionarem com eficácia, é essencial que esses dados sejam gerenciados corretamente para assegurar que permaneçam válidos, consistentes e seguros.

Os dados da aplicação podem ser disponibilizados ao cliente por meio de uma aplicação de servidor (por exemplo, um Serviço da Web), ou a aplicação pode utilizar seus próprios dados locais. Se os dados forem fornecidos por uma aplicação de servidor, a aplicação Smart Client pode armazenar em cachê os dados para melhorar o desempenho ou permitir o uso offline. Neste caso, você precisa decidir como a aplicação do cliente deve manipular dados antigos em relação ao servidor. Se sua aplicação Smart Client oferece o recurso de modificar dados localmente e as mudanças do cliente devem ser sincronizadas com a aplicação do servidor posteriormente. Neste caso, a decisão vai de como a aplicação do cliente irá lidar com conflitos de dados e como registrar as mudanças que serão enviadas ao servidor.

É preciso considerar cuidadosamente esses e muitos outros assuntos ao projetar sua aplicação Smart Client. Este capítulo examina as muitas considerações para a manipulação de dados no cliente, incluindo:

- Tipos de dados
- Cachê de dados
- Simultaneidade de dados
- Utilização dos datasets ADO.NET para gerenciar dados
- Vinculação de dados no Windows Forms

Muitos assuntos relacionados à manipulação de dados não serão abordados neste capítulo. Particularmente, temas sobre segurança na manipulação de dados serão discutidos no capítulo 5, "Considerações de segurança," e considerações offline serão discutidas no capítulo 4, "Smart Clients Conectados Ocasionalmente."

### Tipos de Dados

Smart Clients geralmente têm que administrar dois tipos de dados:

- Dados de referência somente para leitura;
- Dados temporários.

Geralmente, esses tipos de dados necessitam ser manipulados de formas distintas, portanto é necessário examinar cada uma dessas formas detalhadamente.

## Dados de Referência Somente para Leitura

Dados de referência somente para leitura não são alterados pelo cliente e são utilizados pelo cliente para propósitos de referência. Portanto, do ponto de vista do cliente, os dados são somente para leitura e o cliente não efetua operações de atualização, inserção ou exclusão. Dados de referência somente para leitura são prontamente armazenados em cachê no cliente. Dados de referência têm certas utilidades em uma aplicação Smart Client, incluindo:

- **Oferecer referência estática ou lookup data.** Exemplos incluem informações de produto, lista de preços, opções de remessa e preços.
- **Suporte a validação de dados, permitindo que dados adicionados pelo usuário tenham sua correção verificada.** Um exemplo é a verificação de entrada de dados feita em um planejamento de entregas.
- **Auxílio na comunicação com serviços remotos.** Um exemplo é converter uma seleção de usuário em um ID de produto localmente e depois enviar a informação para um Serviço da Web.
- **Apresentação de dados.** Exemplos incluem a apresentação de texto de ajuda ou rótulos de interface.

Ao armazenar e utilizar dados de referência no cliente, você pode reduzir a quantidade de dados que necessitam ser transportados do cliente ao servidor, melhora o desempenho de sua aplicação, ajuda a permitir mais capacidade offline e oferece validação de dados rapidamente, aumentando o aproveitamento de sua aplicação.

Embora dados de referência somente para leitura não possam ser mudados pelo cliente, eles podem ser alterados no servidor (por um administrador ou supervisor, por exemplo). Você precisa determinar uma estratégia para atualização do cliente quando as mudanças de dados ocorrerem. Essa estratégia pode transferir as mudanças para o cliente quando uma mudança ocorrer ou trazer as mudanças do servidor em intervalos determinados ou antes de certas ações no cliente. Entretanto, como os dados são somente para leitura no cliente, você não precisa registrar as mudanças nele. Isso simplifica o modo como dados de referência somente para leitura precisam ser manipulados.

## Dados Temporários

Dados temporários podem ser modificados tanto no cliente quanto no servidor. De modo geral, dados temporários mudam como resultado direto ou indireto das entradas e manipulações do usuário. Neste caso, mudanças feitas tanto no cliente como no servidor precisam ser sincronizadas em algum momento. Esse tipo de dados tem um grande número de utilidades em um Smart Client, incluindo:

- **Inclusão de novas informações.** Exemplos englobam a inclusão de transações bancárias ou detalhes de consumidores.
- **Modificação de informação existente.** Um exemplo é a atualização de detalhes dos consumidores.
- **Apagar informação existente.** Um exemplo é a remoção de um consumidor de um banco de dados.

Um dos aspectos mais desafiadores ao lidar com dados temporários em Smart Clients é o fato de que ele geralmente pode ser modificado em diversos clientes ao mesmo tempo. Este problema é acentuado quando os dados são muito voláteis, porque as mudanças tendem a entrar em conflito umas com as outras.

É preciso registrar qualquer mudança de cliente realizada em dados temporários. Até que os dados sejam sincronizados com o servidor e qualquer conflito seja resolvido, não se deve considerar dados temporários como confirmados. Deve-se ter bastante cuidado para não contar com dados que não tenham sido confirmados na hora de tomada de decisões importantes ou utilizá-los com base para outras mudanças locais sem considerar cuidadosamente como a consistência de dados pode ser garantida mesmo em caso de falha durante a sincronização.

Para mais detalhes sobre o assunto manipulação de dados offline e como manipular a sincronização de dados, consulte o Capítulo 4, "Smart Clients Conectados Ocasionalmente."

## Colocando Dados em Cache

Smart Clients freqüentemente realizam cache de dados localmente, seja de referência somente para leitura ou dados temporários. Colocar dados em cache tem o potencial melhorar o desempenho da aplicação e fornecer os dados necessários para trabalho offline. Entretanto, você necessita considerar com cuidado que dados deve ser colocados em cache no cliente, como esses dados devem ser controlados, e no contexto em que esses dados podem ser usados.

Para permitir que os dados sejam colocados em cache, sua aplicação Smart Client deverá implementar alguns formulários de infra-estrutura de cache que possa manejá-los de forma transparente. Sua infra-estrutura de cache deve incluir um ou mais dos seguintes mecanismos:

- **Short-term data caching.** Colocar dados em cache na memória é bom para o desempenho mas não é persistente, assim que você pode necessitar puxar dados da fonte quando a aplicação rodada novamente. Agir dessa forma pode prevenir que sua aplicação opere quando offline.
- **Long-term data caching.** Colocar dados em cache em um meio persistente, tal como um armazenamento isolado ou um sistema de local de arquivos, permite que se use a aplicação quando não há nenhuma conectividade ao usuário. Você pode escolher combinar o armazenamento a longo prazo com o armazenamento a curto prazo para melhorar a performance.

Apesar dos mecanismos de cache que você adota, você deve assegurar que somente dados a que o usuário tem o acesso estejam disponíveis no cliente. Também, os dados sensíveis colocados em cache no cliente requerem a manipulação cuidadosa para assegurar que sejam mantidos seguros. Conseqüentemente, você pode necessitar codificar os dados enquanto são transferidos ao cliente e enquanto são armazenados no cliente. Para mais informações, ver "Manipulando Dados Sensíveis" no Capítulo 5, "Considerações de Segurança."

Conforme você arquiteta sua aplicação Smart Client para suportar dados em cache, deve ser considerado fornecer um mecanismo a seu cliente para pedir dados recentes, não obstante o estado do cache. Isto significa que você pode estar certo de que a aplicação está pronta para executar transações novas sem usar dados antigos. Você também pode configurar seu cliente a dados recém trazidos que possam suavizar o risco de estar offline quando os dados em cache expirarem.

Sempre que possível, você deve associar algum formulário do metadado com os dados para permitir que o cliente os controle de maneira inteligente. Tal metadado pode ser usado para especificar a identidade de dados e todos os confinamentos ou comportamentos desejados associados com os dados. Sua infra-estrutura de cache deve consumir este metadado e usá-lo para manejar apropriadamente os dados em cache.



Todos os dados colocados em cache no cliente devem ser excepcionalmente identificáveis (por exemplo, através de um número de versão ou de um time stamp), de modo que se possa identificar corretamente ao determinar a necessidade de ser atualizado. Sua infra-estrutura de cache pode então pedir ao servidor se os dados são atualmente válidos e também determinar se alguma atualização é necessária.

O metadado também pode ser usado para especificar as restrições ou os comportamentos que se relacionam ao uso e a manipulação dos dados em cache. Os exemplos incluem:

- **Restrição temporal.** Estas restrições especificam o tempo ou a escala da data em que os dados em cache podem ser usados. Quando os dados se tornam antigos ou expiram, podem ser excluídos do cache ou automaticamente atualizados obtendo os últimos dados do usuário. Em alguns casos, pode ser apropriado deixar o cliente usar dados de referência separado e mapeá-los com dados atualizados quando são sincronizados com o servidor.
- **Restrições Geográficas.** Alguns dados podem ser apropriados somente para uma região particular. Por exemplo, você pode ter listas de preço diferentes para posições diferentes. Sua infra-estrutura de cache pode ser usada para alcançar e armazenar dados em uma base de por-local.
- **Exigências da segurança.** Os dados que são pretendidos especificamente para um usuário particular podem ser codificados para assegurar que somente o usuário apropriado possa acessá-lo. Neste caso, os dados são fornecidos já codificados, e o usuário tem que fornecer as credenciais à infra-estrutura de cache para permitir que os dados sejam decodificados.
- **Regras de Negócios.** Você pode ter regras de negócios associadas com seus dados sem cache que ditam como eles devem ser usados. Por exemplo, sua infra-estrutura de cache pode fazer um exame na consideração do papel do usuário para determinar como os dados serão fornecidos e como serão manejados.

O metadado associado com os dados permite a sua infra-estrutura de cache assegurar apropriadamente os dados de modo que sua aplicação não tenha que se preocupar com a colocação de dados em cache ou detalhes da execução. Você pode passar o metadado associado com os dados referência dentro do próprio dado, ou usar um mecanismo out-of-band. O mecanismo exato usado para transportar o metadado ao cliente depende de como sua aplicação se comunica com os serviços de rede. Quando usar Serviços da Web, usando encabeçamentos do SOAP para comunicar o metadado com cliente é uma boa solução.

As diferenças entre os dados de referência somente de leitura e dos dados temporários às vezes significa a necessidade de usar dois caches, um para dados de referência e outro para dados temporários. Os dados de referência são de leitura apenas no cliente e não necessitam ser sincronizados com o servidor, mas necessitam ser atualizados ocasionalmente para refletir todas as mudanças e atualizações realizadas no servidor.

Os dados temporários podem ser alterados tanto no cliente como no servidor. Com os dados no cache sendo atualizado algumas vezes no cliente, outras no servidor, e às vezes em ambos, todas as mudanças feitas nos dados do cliente necessitam ser sincronizadas com o servidor em algum ponto. Se os dados mudarem no servidor enquanto isso, ocorre um conflito de dados e haverá a necessidade de uma manipulação apropriada. Para ajudar a assegurar que a consistência dos dados seja mantida, e evitar o uso de dados impróprios, deve-se ter cuidado em manter o controle de todas as mudanças que forem realizadas aos dados temporários no cliente. Tais mudanças são descompromissadas ou tentam até que sejam sincronizadas ou confirmadas com sucesso pelo servidor.

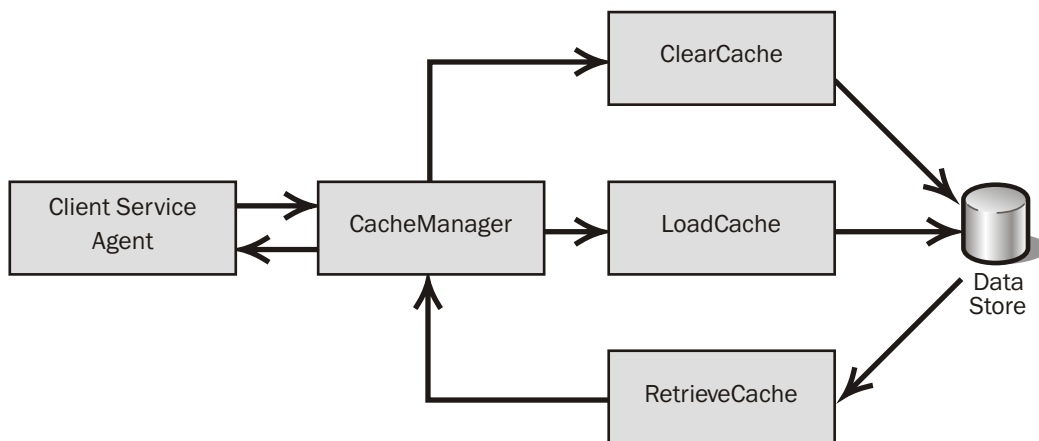
Você deve projetar sua aplicação Smart Client de modo que ela possa diferenciar os dados que foram sincronizados com sucesso com o servidor e os dados que ainda são tentativas. Essa distinção ajuda sua aplicação a detectar mais facilmente e manejar conflitos de dados. Também, você pode querer

restringir a aplicação ou o usuário de tomar decisões importantes ou de iniciar as ações importantes baseadas em dados de tentativa. Tais dados não devem ser confiados sobre até que estejam sincronizados com o usuário. Usando uma infra-estrutura de cache apropriada, você pode manter-se a par de dados confirmados e de tentativa.

## O Caching Application Block

O Caching Application Block é uma extensão do Microsoft® .NET Framework que permite que os desenvolvedores facilmente armazenem em cache os dados dos fornecedores de serviço. Ele foi construído e projetado para encapsular as práticas recomendadas pela Microsoft para colocar em cache as aplicações .NET Framework, conforme descrito em Caching Architecture Guide for .NET Framework Applications em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/CachingArch.asp>.

A arquitetura completa do bloco caching pode ser vista na Figura 2.1.



**Figura 2.1**

*Caching block workflow*

O fluxo de trabalho de caching consiste dos seguintes passos:

1. Um cliente ou agente de serviço faz uma solicitação de itens colocados em cache para o **CacheManager**.
2. Se o item já tiver sido armazenado em cache, o **CacheManager** restaura o item do armazenamento e o retorna como um objeto **CachedItem**. Se o item ainda não foi armazenado em cache, o cliente é notificado.
3. Após restaurar dados não armazenados em cache de um provedor de serviço, o cliente envia os dados para o **CacheManager**. O **CacheManager** adiciona uma assinatura (ou seja, metadado) como uma chave, validade ou prioridade ao item e o carrega no cache.
4. O **CacheService** monitora o tempo de vida dos **CachedItems**. Quando termina a validade de um **CachedItem**, o **CacheService** o remove e, opcionalmente, chama um callback delegado.
5. O **CacheService** também pode eliminar todos os itens do cache.

O caching block oferece diversas opções de validade de caching são descritos na Tabela 2.1.

**Tabela 2.1: Opções de Expiração do Caching**

Classe	Descrição
AbsoluteTime	Utilizada para ajustar o tempo absoluto de expiração.
ExtendedFormatTime	Utilizado para ajustar uma expiração baseada em uma expressão (como a cada minuto ou a cada segunda-feira).
FileDependency	Utilizado para ajustar a expiração para quando um arquivo é alterado.
SlidingTime	Utilizado para ajustar o tempo de vida para um item especificando uma expiração baseada em quando foi acessado pela última vez.

Os mecanismos de armazenamento abaixo estão disponíveis para o caching block:

- **Memory-mapped file (MMF).** MMFs são mais apropriados para um cenário de cache de alto desempenho com base no cliente. Você pode utilizar os MMFs para desenvolver um cache que pode ser compartilhado em muitos domínios de aplicação e processos dentro do mesmo computador. O .NET Framework não oferece suporte a MMFs, portanto qualquer implementação de um cache MMF é executada como código não gerenciado e não se beneficia de qualquer recurso do .NET Framework, incluindo recursos de gerenciamento de memória (como verificação de lixeira) e recursos de segurança (como segurança de acesso de código).
- **Objeto Singleton.** Um objeto Singleton .NET pode ser usado para efetuar cache de dados que podem ser compartilhados através de processos em um ou muitos computadores. Isso é feito com a implementação de um serviço de cache utilizando um objeto singleton que sirva a múltiplos clientes pelo .NET remoto. O cache Singleton é simples de ser implantado, mas não tem o desempenho e a escalabilidade oferecida pelas soluções baseadas no Microsoft SQL Server™.
- **Banco de Dados Microsoft SQL Server 2000.** O armazenamento SQL Server 2000 é mais apropriado para uma aplicação que necessite de alta durabilidade ou quando for necessário efetuar cache de uma grande quantidade de dados. Como o serviço de cache precisa acessar o SQL Server através de uma rede e os dados são restaurados utilizando consultas de banco de dados, o acesso aos dados é relativamente lento.
- **Microsoft SQL Server Desktop Engine (MSDE).** O MSDE é um banco de dados leve e uma alternativa ao SQL Server 2000. Ele oferece recursos de confiabilidade e segurança, mas possui uma quantidade de cliente menor que o SQL Server, necessitando de menos configurações. Como o MSDE oferece suporte ao SQL, os desenvolvedores também ganham muito com o poder de um banco de dados. Você pode migrar um banco de dados MSDE para um banco de dados SQL Server caso seja necessário.

## Simultaneidade de Dados

Como mencionado anteriormente, um problema na utilização de Smart Clients é que mudanças nos dados que acontecem no servidor podem ocorrer antes que mudanças feitas pelo cliente sejam sincronizadas. Você precisa de um mecanismo que assegure que, quando os dados forem sincronizados, qualquer conflito de dados seja manipulado adequadamente e que os dados resultantes sejam consistentes e corretos. A atualização de dados por mais de um cliente é conhecida como simultaneidade de dados.

Existem duas abordagens que podem ser utilizadas para lidar com um caso de simultaneidade de dados:

- **Simultaneidade Pessimista.** A simultaneidade pessimista permite que um cliente mantenha um bloqueio de dados para evitar que qualquer outro cliente modifique os dados até que as mudanças do primeiro cliente tenham sido concluídas. Nesses casos, se outro cliente tentar modificar os dados, a tentativa falhará ou será bloqueada até que o responsável libere o bloqueio.

A simultaneidade pessimista pode ser problemática porque um único usuário ou cliente pode manter essa trava por um período significativo, possivelmente sem saber. Portanto, a trava pode impedir que recursos importantes, como linhas de bancos de dados ou arquivos sejam liberados de modo oportuno, o que pode afetar seriamente a escalabilidade e a aproveitamento da aplicação. No entanto, a simultaneidade pessimista pode ser apropriada quando é preciso ter controle total sobre as mudanças feitas a recursos importantes. Observe que ela não pode ser utilizada se seus clientes trabalham offline, porque não será possível bloquear os dados.

- **Simultaneidade Otimista.** A simultaneidade otimista não bloqueia os dados. Para decidir se uma atualização é realmente necessária, os dados originais podem ser enviados juntamente com as solicitações de atualização e os dados modificados. Os dados originais são verificados em relação aos dados atuais para procurar por atualizações. Se os dados originais e os atuais forem iguais, a atualização é executada; do contrário, a solicitação é negada, produzindo uma falha otimista. Para otimizar esse processo, pode-se utilizar um timestamp ou um contador de atualizações nos dados em vez de enviar os originais e nesse caso, somente o timestamp ou o contador precisarão ser verificados.

A simultaneidade otimista oferece um bom mecanismo para atualizar dados mestres que não mudam com frequência, como telefones ou endereços de consumidores. A simultaneidade otimista permite que todos leiam os dados e em situações em que as atualizações são menos prováveis que operações de leitura, o risco de uma falha otimista pode ser aceitável. A simultaneidade otimista pode não ser adequada em situações nas quais os dados são modificados com frequência ou nas quais as atualizações otimistas têm mais probabilidade de falhar com maior frequência.

Na maioria dos cenários Smart Client, incluindo aqueles nos quais os clientes trabalham offline, a simultaneidade otimista é a abordagem correta, por permitir que múltiplos clientes trabalhem nos dados ao mesmo tempo sem bloqueá-los desnecessariamente, afetando todos os outros clientes.

Para mais informações sobre simultaneidade pessimista e otimista, consulte "Optimistic Concurrency" no .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoptimisticconcurrency.asp>.

## Utilizando DataSets ADO.NET para Gerenciar Dados

Um **DataSet** é um objeto que representa uma ou mais tabelas de banco de dados relacional. Os Datasets armazenam dados em um cache desconectado. A estrutura de um dataset é semelhante à de um banco de dados relacional: Eles expõem um modelo de objeto hierárquico de tabelas, linhas e colunas. Além disso, eles contêm relacionamentos e restrições definidos para o dataset.

Um **DataSet** ADO.NET contém uma coleção de zero ou mais tabelas representadas por objetos **DataTable**. Uma **DataTable** é definido no espaço de nome **System.Data** e representa uma tabela única de dados residentes na memória e contém um conjunto de colunas representado por um **DataColumnCollection** e restrições representadas por um **ConstraintCollection** que juntos definem o esquema da tabela. Uma **DataTable** também contém uma coleção de linhas representadas pelo **DataRowCollection** que contém os dados da tabela. Juntamente com seu estado atual, um **DataRow** retém tanto a versão atual quanto a original para identificar mudanças nos valores armazenados na linha. Os **Datasets** podem ser strongly typed ou untyped. Um **DataSet** typed herda da classe básica do DataSet, mas adiciona funcionalidade de linguagem strong typed, permitindo que os usuários

acessem conteúdos de modo programático com strongly typed. Os dois tipos podem ser utilizados na construção de aplicações. No entanto, o sistema de desenvolvimento Microsoft Visual Studio® suporta melhor typed datasets, e eles tornam a programação com datasets mais fácil e com menor propensão a erros.

Datasets são particularmente úteis em ambiente Smart Client porque oferecem funcionalidade para ajudar os clientes a trabalhar com dados enquanto estão offline. Eles podem registrar mudanças locais nos dados, o que ajuda a sincronizar os dados com o servidor e solucionar conflitos, podendo ser utilizados para mesclar dados de diferentes origens.

Para mais informações sobre trabalho com datasets, consulte "Introduction to Datasets" em Visual Basic and Visual C# Concepts em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconDataSets.asp>.

## Mesclando Dados com Datasets

Os Datasets possuem a habilidade de mesclar os conteúdos de objeto do **DataSet**, **DataTable**, ou **DataRow** em datasets existentes. Essa funcionalidade é especialmente útil para registrar mudanças no cliente e mesclá-las com conteúdos atualizados do servidor.

A Figura 2.2 mostra um Smart Client solicitando uma atualização do Serviço da Web e os novos dados sendo devolvidos como objeto de transferência de dados (DTO). Um DTO é um padrão corporativo que permite que se agrupe em um objeto todos os dados necessários para comunicar-se com um Serviço da Web. Utilizar um DTO geralmente significa que você pode fazer uma só chamada para um Serviço da Web em vez de diversas chamadas.



**Figura 2.2**

*Fusão de dados no cliente usando datasets*

Neste exemplo, quando o DTO retorna para o cliente, ele é usado para criar um novo dataset localmente.

---

**Observação:** Após uma operação de mescla, o ADO.NET não muda automaticamente o estado de alterado para inalterado. Portanto, após mesclar os novos datasets com o dataset de cliente local, é preciso invocar o método `AcceptChanges` em seu dataset para reconfigurar a propriedade do `RowState` para inalterado.

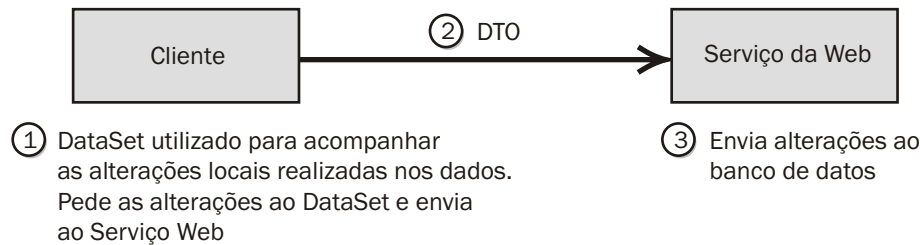
---

Para mais informações sobre a utilização de datasets, consulte "Merging DataSet Contents" no .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconmergingdatasetcontents.asp>.

## Aumentando o Desempenho dos Datasets

Os Datasets freqüentemente contêm um grande número de dados que, se transmitidos pela rede, podem resultar em problemas de desempenho. Felizmente, com o ADO.NET DataSets, você pode usar o método `GetChanges` em seus datasets para assegurar que somente os dados modificados em um dataset comuniquem-se entre o cliente e o servidor, acomodando os dados em um DTO. Em seguida, os dados são mesclados com o dataset em seu destino.

Figura 2.3 mostra um Smart Client que faz mudanças em dados locais e utiliza o método `GetChanges` em um dataset para submeter somente os dados modificados ao servidor. Os dados são transferidos a um DTO por motivos de desempenho.

**Figura2.3**

Usando um DTO para melhorar o desempenho

O método `GetChanges` pode ser usado por aplicações Smart Client que precisam ficar offline. Quando a aplicação estiver on-line novamente, você pode utilizar o método **`GetChanges`** para determinar quais informações mudaram e, em seguida, gerar um DTO para se comunicar com o Serviço da Web e assegurar que as mudanças sejam enviadas ao banco de dados.

## Vinculação de Dados no Windows Forms

A vinculação de dados no Windows Forms permite que conecte a interface de usuário de sua aplicação aos dados subjacentes da aplicação. A vinculação de dados no Windows Forms suporta vinculação bidirecional para que se possa vincular uma estrutura de dados com a interface de usuário, mostrar os valores atuais de dados ao usuário, permitir que o usuário edite esses dados e depois atualizar os dados subjacente automaticamente, utilizando os valores inseridos pelo usuário.

Você pode usar a vinculação de dados no Windows Forms para conectar praticamente qualquer estrutura de dados ou objeto a qualquer propriedade dos controles da interface do usuário. Você pode vincular um único item de dados a uma propriedade única de controle ou de dados mais complexos (por exemplo, uma coleção de itens de dados ou uma tabela de banco de dados) ao controle para que ele possa mostrar todos os dados em data grid ou list box.

---

**Observação:** Você pode vincular qualquer objeto que suporte uma ou mais propriedades públicas. É possível vincular apenas as propriedades públicas da sua classe e não a membros públicos.

---

A vinculação de dados no Windows Forms permite que você ofereça uma interface de usuário flexível e data-driven com suas aplicações. Você pode utilizar a vinculação de dados para oferecer controle personalizado sobre a aparência de sua interface de usuário (por exemplo, vinculando a propriedades de controle como cor de segundo plano ou de primeiro plano, tamanho, imagem ou ícone).

A vinculação de dados possui muitos usos. Por exemplo:

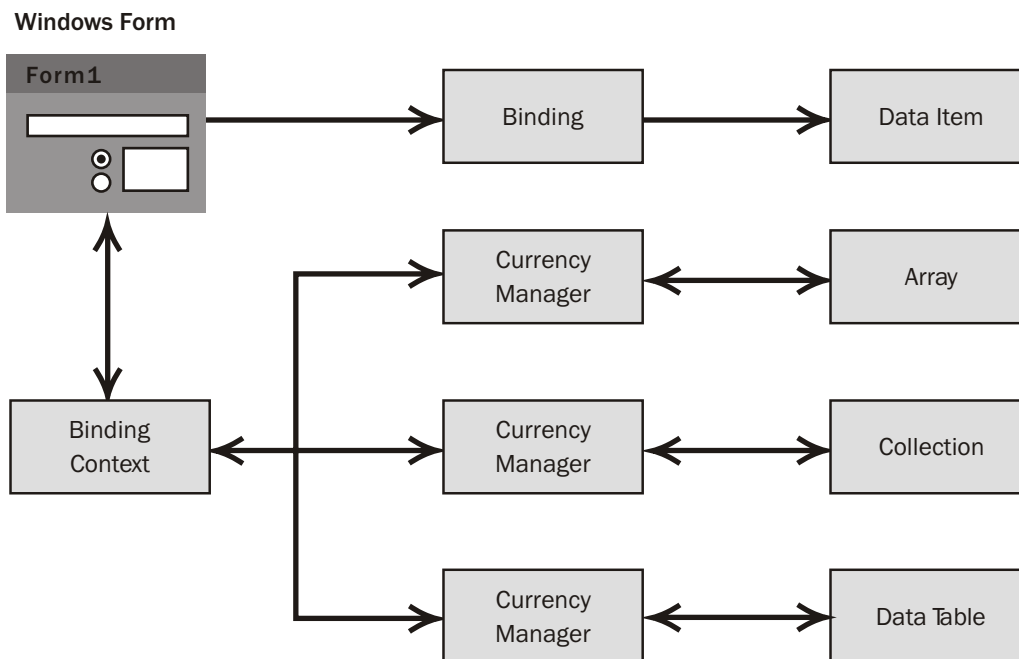
- Mostrar dados somente de leitura aos usuários;
- Permitir que usuários atualizem dados a partir da interface de usuário;
- Oferecer visualização detalhada dos dados;
- Permitir que os usuários explorem itens complexos de dados relacionados;
- Oferecer funcionalidade lookup na tabela, permitindo que a interface de usuário conecte-se a nomes de exibição de usuários amigáveis.

Esta sessão examina algumas características da vinculação de dados e discute alguns dos atributos da vinculação de dados que você freqüentemente precisará implementar em uma aplicação Smart Client.

Para obter informações mais aprofundadas sobre vinculação, consulte "Windows Forms Data Binding and Objects" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnadvnet/html/vbnet02252003.asp>.

## Arquitetura de Vinculação de Dados no Windows Forms

A vinculação de dados no Windows Forms oferece uma infra-estrutura flexível para conectar dados bidirecionalmente à interface de usuário. A Figura 2.4 mostra uma representação esquemática da arquitetura geral de vinculação de dados no Windows Forms.



**Figura 2.4**

*Arquitetura de vinculação de dados no Windows Forms*

A vinculação de dados no Windows Forms utiliza os seguintes objetos:

- **Origem de Dados.** Fontes de dados são objetos que contêm dados que serão vinculados à interface de usuário. Fornecedores de dados podem ser qualquer objeto que tenha propriedades públicas, um conjunto ou uma coleção que suporte a interface `ICollection` ou uma instância de uma classe de dados complexa (por exemplo, **DataSet** ou **DataTable**).
- **CurrencyManager.** O objeto **CurrencyManager** registra a posição atual de dados dentro de um conjunto, coleção ou tabela que é vinculado à interface de usuário. O **CurrencyManager** permite que você vincule uma coleção de dados a uma interface de usuário e navegue por ela, atualizando a interface de usuário de modo a refletir o item selecionado atualmente dentro da coleção.
- **PropertyManager.** O objeto **PropertyManager** é responsável pela manutenção da propriedade atual de um objeto vinculado a um controle. Tanto a classe do **PropertyManager** quanto a classe **CurrencyManager** herdam de uma classe de base comum, **BindingManagerBase**. Todos os fornecedores de dados vinculados a um controle para ter um objeto associado **CurrencyManager** ou **PropertyManager**.
- **BindingContext.** Cada Windows Form tem um objeto **BindingContext** padrão que registra todos os objetos **CurrencyManager** e **PropertyManager** no formulário. O objeto **BindingContext** permite que você restaure facilmente os objetos **CurrencyManager** ou **PropertyManager** para uma origem de dados específica. Você pode designar um objeto **BindingContext** específico para um controle de

container (como um **GroupBox**, **Panel** ou **TabControl**) que contenha controles de data-bound. Isso permite que cada parte de seu formulário seja gerenciada por seu próprio objeto **CurrencyManager** ou **PropertyManager**.

- **Binding.** Os objetos **Binding** são usados para criar e manter uma vinculação simples entre uma única propriedade de um controle e a propriedade de outro objeto ou a do objeto atual em uma lista de objetos.

## Vinculação de Dados a Controles do Windows Forms

Há diversas propriedades e métodos que se pode utilizar para vincular a controles específicos do Windows Forms. A Tabela 2.2 mostra alguns dos mais importantes.

**Tabela 2.2: Propriedades e Métodos para Vinculação de Dados a Controles do Windows Forms**

Propriedade ou Método	Controle do Windows Forms	Descrição
Propriedade DataSource	ListControls (por exemplo: ListBox ou Combo Box), DataGrid Control	Permite especificar o objeto provedor dos dados ( <i>data provider</i> ) para ser a ligação o controle da interface do usuário.
Propriedade DisplayMember	ListControls	Permite especificar o membro do provedor de dados ( <i>data provider</i> ) que será apresentado ao usuário.
Propriedade ValueMember	ListControls	Permite especificar o valor associado com o apresentado para um uso interno da sua aplicação.
Propriedade DataMember	DataGrid control	Se o <b>DataSource</b> conter mais de uma fonte de dados (por exemplo, se for especificado que o <b>DataSet</b> contenha múltiplas tabelas), utilizarem a propriedade do <b>DataMember</b> para especificar um para ser a ligação com o <i>grid</i> (veja nota abaixo).
Método SetDataBinding	DataGrid control	Permite que se reinicie o método DataSource em tempo de execução.

**Note:** Se o **DataSource** é um **DataTable**, **DataView**, **Collection** ou **Array**, o ajuste da propriedade do **DataMember** não é requerida.

Pode-se também utilizar a propriedade de coleção do **DataBindings** disponíveis em todos os objetos de controle do Windows Forms para incluir explicitamente objetos **Binding** em qualquer objeto de controle. Os objetos **Binding** são usados para vincular uma única propriedade no controle a um único membro de dados do provedor de dados. O exemplo de código abaixo adiciona um vínculo entre a propriedade **Text** de um controle de caixa de texto para o nome do consumidor na tabela de consumidores de um dataset.

```
textBox1.DataBindings.Add(
    new Binding( "Text", dataset, "customers.customerName" ) );
```

Quando você cria uma instância **Binding** com um construtor **Binding**, você deve especificar o nome da propriedade de controle a qual será vinculada, a origem de dados e o caminho de navegação que leva a uma lista ou propriedade na origem de dados. O caminho de navegação pode ser uma cadeia vazia, um único nome de propriedade ou uma hierarquia de nomes delimitada por pontos. Você pode utilizar uma caminho de navegação hierárquico para navegar através de tabelas de dados e relações em um objeto **DataSet** ou através de um modelo de objeto onde as propriedades de um objeto retornam



instâncias a outros objetos. Se você configurar o caminho de navegação para uma cadeia vazia, o método **ToString** é chamado no objeto subjacente da origem de dados.

---

**Observação:** Se uma propriedade é somente para leitura (ou seja, o objeto não suporta um conjunto de operações para essa propriedade), a vinculação de dados não torna, por padrão, o controle do Windows Forms vinculado somente para leitura. Isso pode deixar o usuário confuso, pois ele poderá editar o valor na interface do usuário, mas o valor no objeto vinculado não será atualizado. Portanto, assegure que os sinalizadores de "somente leitura" estejam definidas para true em todos os controles do Windows Form que estão vinculados a propriedades de "somente leitura".

---

## Vinculando Controles aos DataSets

Freqüentemente, é útil vincular controles a datasets. Isso possibilita exibir os dados dataset em um data grid, e isso permite que o usuário facilmente atualize os dados. Você pode vincular um controle data grid a um **DataSet** usando o seguinte código.

```
DataSet newDataSet = webServiceProxy.GetDataSet();  
this.dataGrid.SetDataBinding( newDataSet, "tableName" );
```

Algumas vezes é necessário substituir os conteúdos de seu dataset depois de todos os vínculos com seus controles terem sido estabelecidos. No entanto, quando você substitui sets existentes com sets novos, todos os vínculos permanecem com os datasets antigos.

Em vez de recriar manualmente os vínculos de dados com a nova fonte, você pode utilizar o método **Merge** da classe **DataSet** para trazer os dados do novo data set ao existente, como mostrado no exemplo de código abaixo.

```
DataSet newDataSet = myService.GetDataSet();  
this.dataSet1.Clear();  
this.dataSet1.Merge( newDataSet );
```

---

**Observação:** Para evitar questões de threading, deve-se atualizar apenas objetos de dados vinculados no encadeamento UI. Para mais informações, consulte Capítulo 6, "Utilizando Threads Múltiplos."

---

## Navegando Através de uma Coleção de Dados

Se sua origem de dados possui uma coleção de itens, você pode vincular a coleção de dados a seus controles do Windows Forms e navegar através da coleção de dados, um item de cada vez. A interface de usuário é atualizada automaticamente para refletir o item atual na coleção.

Você pode vincular a qualquer objeto de coleção que suporte a interface **IList**. Quando você vincula a uma coleção de objetos, é possível permitir ao usuário navegar através de cada item da coleção, atualizando-a automaticamente para cada item. Muitas das coleções e classes de dados complexas oferecidas pelo .NET Framework já suportam a interface **IList**, para que se possa facilmente vincular conjuntos ou dados complexos como data rows ou data views. Por exemplo, qualquer objeto de array que seja uma instância da classe **System.Array** implementa a interface **IList** como padrão e portanto pode ser vinculado à interface de usuário. Muitos dos objetos ADO.NET também suportam a interface **IList** ou uma derivação, permitindo que esses objetos sejam facilmente vinculados também. Por exemplo, todas as classes **DataManager**, **DataSet**, **DataTable**, **DataRowView** e **DataRowColumn** suportam vinculação de dados dessa forma.

Origens de dados que implementam a interface **IList** são gerenciadas pelo objeto **CurrencyManager**. Este objeto mantém um índice na coleção de dados através da sua propriedade **Position**. O índice é usado para garantir que todos os controles vinculados à fonte de dados leiam e gravem no mesmo item da coleção de dados.

Se seu formulário contém controles vinculados a múltiplas origens de dados, ele terá múltiplos objetos **CurrencyManager**, um para cada origem de dados distinta. O objeto **BindingContext** fornece acesso fácil a todos os objetos **CurrencyManager** do formulário. O seguinte exemplo de código mostra como incrementar a posição atual dentro de uma coleção de consumidores.

```
this.BindingContext[ dataset, "customers" ].Position += 1;
```

Deve-se utilizar a propriedade **Count** no objeto **CurrencyManager**, como demonstrado no código abaixo, para assegurar que uma posição inválida não seja definida.

```
if ( this.BindingContext[ dataset, "customer" ].Position <
    ( this.BindingContext[ dataset, "customer" ].Count - 1 ) )
{
    this.BindingContext[ dataset, "customers" ].Position += 1;
}
```

O objeto **CurrencyManager** também suporta um evento **PositionChanged**. Você pode criar um manipulador para este evento, para que seja possível atualizar sua interface de usuário de modo a refletir a posição de vínculo atual. O seguinte exemplo de código exibe uma etiqueta para visualização da posição atual e o número total de registros.

```
this.BindingContext[ dataset, "customers" ].PositionChanged +=
    new EventHandler( this.BindingPositionChanged );
```

O método **BindingPositionChanged** é implementado da seguinte forma.

```
private void BindingPositionChanged( object sender, System.EventArgs e )
{
    positionLabel.Text = string.Format( "Record {0} of {1}",
        this.BindingContext[dsPubs1, "authors"].Position + 1,
        this.BindingContext[dsPubs1, "authors"].Count );
}
```

## Formatação Personalizada e Conversão Data Type

Você pode fornecer formatação personalizada para dados vinculados a um controle utilizando os eventos **Format** e **Parse** da classe **Binding**. Esses eventos permitem que se controle o modo de exibição de dados na interface de usuário e como os dados serão retirados da interface e examinados, para que os dados subjacentes possam ser atualizados. Esses eventos também podem ser utilizados para converter tipos de dados para que tanto a origem quanto o destino sejam compatíveis.

---

**Observação:** Se o tipo de dados da propriedade vinculada no controle não corresponder ao tipo de dados da origem de dados, será emitida uma exceção. Se você precisar vincular tipos incompatíveis, você deve utilizar os eventos **Format** e **Parse** no objeto **Binding**.

---

O evento **Format** ocorre quando os dados são lidos a partir da origem de dados e exibidos no controle e quando os dados são lidos do controle e usados para atualizar a origem de dados. Quando os dados são lidos a partir da origem de dados, o objeto **Binding** usa o evento **Format** para exibir os dados formatados no controle. Quando os dados são lidos a partir do controle e utilizados para atualizar a origem de dados, o objeto **Binding** examina os dados utilizando o evento **Parse**.

Os eventos **Format** e **Parse** permitem que você crie formatação personalizada para a exibição de dados. Por exemplo, se os dados na tabela forem do tipo **Decimal**, você pode exibi-los no formato de

moeda local configurando a propriedade **Value** do objeto **ConvertEventArgs** para o valor formatado no evento **Format**. Conseqüentemente, você deverá formatar o valor exibido no evento **Parse**.

O exemplo de código abaixo vincula um valor de pedido a uma caixa de texto. Os eventos **Format** e **Parse** são usados para converter entre o string esperado pela caixa de texto e os tipos decimais esperados pela origem dos dados.

```
private void BindControl()
{
    Binding binding = new Binding( "Text", dataset,
    "customers.custToOrders.OrderAmount" );
    // Add the delegates to the event.
    binding.Format += new ConvertEventHandler( DecimalToCurrencyString );
    binding.Parse += new ConvertEventHandler( CurrencyStringToDecimal );
    text1.DataBindings.Add( binding );
}

private void DecimalToCurrencyString( object sender, ConvertEventArgs
cevent )
{
    // The method converts only to string type. Test this using the
    DesiredType.
    if( cevent.DesiredType != typeof( string ) ) return;
    // Use the ToString method to format the value as currency ("c").
    cevent.Value = ((decimal)cevent.Value).ToString( "c" );
}

private void CurrencyStringToDecimal( object sender, ConvertEventArgs
cevent )
{
    // The method converts back to decimal type only.
    if( cevent.DesiredType != typeof( decimal ) ) return;
    // Converts the string back to decimal using the static Parse
    method.
    cevent.Value = Decimal.Parse( cevent.Value.ToString(),
    NumberStyles.Currency, null );
}
```

## Utilizando o Padrão Model View Controller para Implementar Validação de Dados

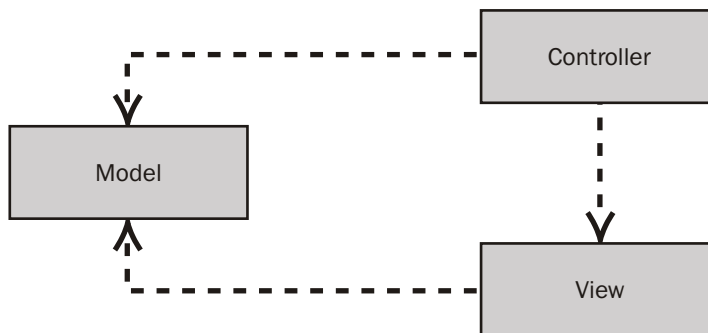
Vincular uma estrutura de dados a um elemento de interface de usuário permite ao usuário editar dados e assegura que essas mudanças sejam gravadas novamente na estrutura de dados subjacente. Frequentemente, você precisará verificar as mudanças feitas nos dados pelos usuários para assegurar-se de que os dados inseridos são válidos.

Os eventos **Format** e **Parse** descritos na sessão anterior oferecem uma maneira de interceptar as mudanças feitas pelos usuários nos dados, para que sua validação possa ser verificada. No entanto, esta abordagem requer que a lógica de validação de dados seja implementada juntamente com o código de formatação personalizada, geralmente no nível do formulário. Implementar essas duas responsabilidades juntas nos manipuladores de evento podem tornar seu código difícil de entender e manter.

Uma abordagem mais correta é projetar seu código para que ele utilize o padrão Model-View-Controller (MVC). O padrão oferece separação natural das várias responsabilidades envolvidas na edição e alteração de dados através da vinculação de dados. Você deve implementar a formatação personalizada dentro do formulário responsável pela apresentação de dados em um determinado formato e associar as regras de validação com os dados em si, para que essas regras possam ser reutilizadas de diferentes maneiras.

No padrão MVC, os dados são condensados em um objeto modelo. O objeto visualizado é o *controle do Windows Forms* ao qual o dado está vinculado. Todas as alterações a este modelo são manipuladas por um objeto controlador intermediário, responsável pelo fornecimento de acesso aos dados e pelo controle de quaisquer mudanças feitas nos dados através do objeto visualizado. O controlador de objeto oferece uma localização natural para a validação de alterações feitas nos dados e toda lógica de validação de interface de usuário deve ser implementada nesse local.

**A Figura 2.5 ilustra o relacionamento estrutural entre os três objetos de um padrão MVC.**



**Figure 2.5**

*Objetos no padrão Model-View-Controller*

Utilizar um objeto controlador desse modo possui muitas vantagens. Você pode configurar um controlador genérico para fornecer regras padrão de validação, que podem ser configuradas durante o tempo de execução de acordo com informações contextuais (por exemplo, a função do usuário). Você também pode fornecer diversos objetos controladores, com cada um implementando regras de validação específicas e depois selecionar o objeto apropriado no tempo de execução. De qualquer maneira, como toda lógica de validação está condensada no objeto controlador, os objetos view e modelo não precisam ser alterados.

Além de separação de dados, lógica de validação e controles de interface de usuário, o modelo MVC oferece uma maneira simples de atualizar automaticamente a interface de usuário quando os dados subjacentes são alterados. O objeto controlador é responsável pela notificação da interface de usuário quando alterações de dados acontecem por meio de outras maneiras de programação. A vinculação de dados no Windows Forms que escuta os eventos gerados pelos objetos que são limitados aos controles de modo que interface de usuário possa automaticamente responder às mudanças feitas nos dados subjacentes.

Para implementar atualizações automáticas da interface de usuário, deve ser assegurado de que o controlador implemente um evento de notificação de alteração para cada propriedade que possa ser alterada. Os eventos devem seguir a convenção de nomenclatura **<property>Changed**, em que **<property>** é o nome da propriedade. Por exemplo, se o controlador suporta uma propriedade **Name**, ele deve suportar também um evento **NameChanged**. Se o valor da propriedade nome mudar, esse evento deveria ser disparado para que a vinculação de dados no Windows Forms possa manipulá-lo e atualizar a interface de usuário.

O exemplo de código abaixo define um objeto **Customer**, que implementa uma propriedade **Name**. O objeto **CustomerController** manipula a lógica de validação para um objeto **Customer** e suporta uma propriedade **Name** que, por sua vez, representa a propriedade **Name** no objeto **Customer** subjacente. Este controlador dispara um evento sempre que o nome for alterado.

```
public class Customer
{
    private string _name;
    public Customer( string name ) { _name = name; }
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}

public class CustomerController
{
    private Customer _customer = null;
    public event EventHandler NameChanged;
    public Customer( Customer customer )
    {
        this._customer = customer;
    }
    public string Name
    {
        get { return _customer.Name; }
        set
        {
            // TODO: validate new name to make sure it is valid.
            _customer.Name = value;
            // Notify bound control of change.
            if ( NameChanged != null )
                NameChanged( this, EventArgs.Empty );
        }
    }
}
```

---

**Observação:** Membros da origem de dados de consumidores necessitam ser inicializados quando são declarados. No exemplo anterior, o membro **consumer.Name** precisa ser inicializado em uma cadeia vazia. Isso acontece porque o .NET Framework não tem chance de interagir com o objeto e aplicar a configuração padrão em uma cadeia vazia antes que a vinculação de dados ocorra. Se o membro da origem de dados de consumidores não for inicializado, a tentativa de recuperar um valor de uma variável não inicializada causa uma exceção de tempo de execução.

---

No exemplo de código a seguir, O formulário tem um objeto **TextBox**, **textbox1**, que precisa ser vinculado ao nome do cliente. O código vincula a propriedade **Text** do objeto **TextBox** à propriedade **Name** do controlador.

```
_customer = new Customer( "kelly Blue" );  
_controller = new CustomerController( _customer );  
Binding binding = new Binding( "Text", _controller, "Name" );  
textbox1.DataBindings.Add( binding );
```

Se o nome do consumidor mudar (utilizando a propriedade **Name** no controlador), o evento **NameChanged** é disparado e a caixa de texto é atualizada automaticamente, refletindo o novo valor de nome.

## Atualizando a Interface de Usuário quando os Dados Subjacentes são Alterados

Você pode usar a vinculação de dados do Windows Forms para atualizar automaticamente a interface de usuário quando os dados subjacentes correspondentes são alterados. Isso é feito implementando-se um evento de notificação de mudança no objeto vinculado. Eventos de notificação de mudança são nomeados de acordo com a seguinte convenção:

```
public event EventHandler <propertyName>Changed;
```

Por exemplo, se você vincula uma propriedade do objeto **Name** à interface de usuário e o nome do objeto muda como resultado de outro processamento, você pode atualizar automaticamente a interface de usuário para refletir o novo valor **Name** implementando o evento **NameChanged** no objeto vinculado.

## Sumário

Há muitas considerações diferentes envolvidas na determinação de como manipular dados em seus Smart Clients. Você precisa determinar se e como efetuar cache de seus dados e como lidar com problemas de simultaneidade de dados. Frequentemente você decidirá pela utilização de **datasets** ADO.NET para manipular seus dados e você provavelmente também decidirá tirar vantagem da funcionalidade de vinculação de dados do Windows Forms.

Em muitos casos, dados de referência somente de leitura e dados temporários necessitarão ser tratados de maneira distinta. Como os Smart Clients geralmente utilizam os dois tipos de dados, é necessário determinar a melhor forma de manipular cada categoria de dados em sua aplicação.

# 3

## Conectando-se

Por definição, Smart Clients necessitam se conectar e comunicar com outros recursos e formar parte de uma aplicação distribuída. Esses recursos podem ser processos ou componentes do cliente ou ser recursos de rede, como um Serviço da Web.

Este capítulo examina a natureza da comunicação entre Smart Clients e outros recursos. Ele trata das diferentes tecnologias disponíveis para conexão e uso de recursos em outros processos, componentes ou serviços remotos e discute como escolher entre elas. Finalmente, examina como projetar melhor seu Smart Client para conectar-se aos recursos.

### Sistemas Loosely Coupled e Tightly Coupled

Uma aplicação de cliente pode conectar e utilizar componentes e serviços em outros processos, tanto localmente como na rede, de muitas maneiras diferentes. É interessante categorizar as diferentes abordagens pela quantidade de acoplamento (coupling) existente entre o serviço e o cliente.

Coupling é o grau em que um componente (em um sistema distribuído) depende do outro. A natureza do coupling entre clientes e os serviços com os quais se comunicam pode afetar muitos aspectos do projeto do Smart Client, incluindo a interoperabilidade, recursos offline, desempenho de comunicação com a rede, implementação e considerações de manutenção.

Sistemas Tightly Coupled (frouxamente acoplado) freqüentemente fornecem comunicação direta objeto-a-objeto, com o objeto no cliente recebendo informações detalhadas sobre o objeto remoto. Esse sistema pode evitar atualizações independentes no cliente ou no servidor. Como envolvem comunicação direta objeto-a-objeto, os objetos geralmente interagem com mais freqüência do que em sistemas Loosely Coupled (firmemente acoplado), que podem causar problemas de latência e de desempenho caso os dois objetos estejam em computadores separados e sejam separados por uma conexão de rede.

Sistemas Loosely Coupled são geralmente sistemas baseados em mensagens, sem que o cliente e o serviço remoto tenham conhecimento de como o outro é implementado. Qualquer comunicação entre o cliente e o serviço é ditada pelo esquema da mensagem. Contanto que as mensagens estejam em conformidade com o esquema acordado, a implementação do cliente ou do serviço pode ser alterada segundo necessário sem que exista o temor de interferência sobre o outro.

Os mecanismos de comunicação Loosely Coupled oferecem diversas vantagens sobre os mecanismos Tightly Coupled e ajudam a reduzir a dependência entre o cliente e o serviço remoto. No entanto, Tight Coupling freqüentemente oferece benefícios de desempenho e permitem uma integração mais próxima entre o cliente e o serviço, o que pode ser necessário devido a exigências de segurança ou de transação.

Todos os clientes distribuídos que se comunicam com serviços ou componentes remotos têm algum grau de coupling. Você precisa estar ciente das diferentes características entre as diversas abordagens de Loosely Coupled e Tightly Coupled para que você possa escolher o grau correto de integração para sua aplicação.

## Opções de Comunicação

Quando você projeta a sua aplicação Smart Client, você pode escolher dentre um certo número de métodos para conecta-la a outros recursos, incluindo:

- **Microsoft® .NET Enterprise Services**
- **Microsoft .NET remoting**
- **Microsoft Windows® Message Queuing (também conhecido como MSMQ)**
- **Serviços da Web**

### .NET Enterprise Services

Você pode usar o .NET Enterprise Services para oferecer acesso à infra-estrutura de serviços COM+ de componentes e aplicações de código gerenciado. Componentes .NET contam com o COM+ para fornecer-lhes diversos serviços de componentes, como:

- Suporte Transacional
- Segurança Baseada em Função
- Eventos Loosely Coupled
- Agrupamento de Objetos
- Alinhamento de Componentes
- Ativação Just-in-Time

Um componente .NET que utilize serviços COM+ é conhecido como serviced component. Como os serviced components são hospedados por uma aplicação COM+, eles devem estar acessíveis para essa aplicação. Isso introduz diversos requisitos de registro e configuração:

- A montagem deve derivar-se da classe **ServicedComponent** no espaço de nomes **System.EnterpriseServices**.
- O assembly deve ser strong-named.
- O assembly deve estar no registro do Microsoft Windows.
- Definições de tipo de biblioteca para a montagem devem ser registradas e instaladas em uma aplicação COM+ específica.

Montagens que contenham serviced components configurados como aplicações out-of-process (fora de processo) não precisam ser colocadas no cache de assembly global. As montagens que contêm serviced components configurados como bibliotecas in-process (dentro do processo) não precisam ser colocadas no cache global, a menos que estejam localizados em diretórios diferentes do da aplicação.



Se forem implementadas diversas cópias da mesma versão de um serviced component desta maneira, o catálogo COM+ conterá a configuração global para todas as instâncias desse componente; não é possível configurá-los em uma base por cópia.

O exemplo de código a seguir mostra um componente que requer uma transação e fornece um métodos que grava dados em um banco de dados dentro dessa transação.

```
using System.EnterpriseServices;

[Transaction( TransactionOption.Required )]

public class CustomerAccount : ServicedComponent
{
    [AutoComplete]

    public bool UpdateCustomerName( int customerID, string customerName )
    {
        // Updates the database, no need to call SetComplete.
        // Calls SetComplete automatically if no exception is thrown.

    }
}
```

Geralmente, os Serviced Components podem se registrar dinamicamente na primeira vez em que são executados. Este tipo de registro é conhecido como *lazy registration*. A primeira vez que uma aplicação de código gerenciado tenta criar uma instância para um serviced component, a common language runtime (CLR) registra a montagem e a biblioteca de tipo e configura o catálogo COM+. O registro ocorre somente uma vez para cada versão particular de montagem.

A Lazy Registration permite implementar serviced components utilizando implementação **Xcopy** e trabalhar com serviced components durante o ciclo de desenvolvimento, tendo que registrá-los explicitamente.

A Lazy Registration é o método mais fácil para registrar seus serviced components, mas funciona somente se o processo que os estiver executando tiver privilégios administrativos. Além disso, qualquer montagem marcada como uma aplicação de servidor COM+ requer registro explícito; o registro dinâmico não funciona em clientes não gerenciados que chamam serviced components gerenciados. Nos casos em que o processo que utiliza o serviced component não tenha os privilégios necessários para o registro dinâmico, é necessário registrar a montagem que contenha o serviced component utilizando a ferramenta Regsvcs.exe, fornecida com o .NET Framework.

Tanto a Lazy Registration quanto o Regsvcs.exe requerem permissões administrativas no computador cliente, portanto, se a sua aplicação incluir serviced components, ela não pode ser implementada utilizando implementação no-touch. Para mais detalhes, consulte o Capítulo 7, "Implementação e Atualização das Aplicações Smart Client".

Os Serviced Components podem ser hospedados e acessados de diversas maneiras. Eles podem ser hospedados dentro de uma aplicação ASP.NET e acessados através de HTTP ou do SOAP ou DCOM (a configuração padrão). No entanto, se os serviços COM+ necessitam fluir com o chamado (por exemplo, se você precisa da identidade do usuário ou que uma transação distribuída flua a partir de sua aplicação para o serviced component), o DCOM é a única solução viável.

---

**Observação:** Se você usar o DCOM para comunicar-se com as aplicações COM+, você precisará implementar montagens interop nos computadores clientes, como faria com componentes COM tradicionais.

---

O Enterprise Services tem muitos recursos poderosos de componentes que você pode utilizar em suas aplicações Smart Client. No entantossomente utilizar essas características apenas dentro de um processo único, em um único computador cliente, ou dentro dos limites de um serviço no servidor. O Enterprise Services em geral não é a melhor escolha para comunicação entre uma aplicação Smart Client e serviços localizados em sistemas remotos devido à sua natureza tightly coupled. Utilize Enterprise Services se suas aplicações Smart Client precisarem utilizar serviços COM+ localmente (por exemplo, suporte para transações, união de objetos ou segurança baseada em função).

Para mais informações sobre Enterprise Services, consulte "Writing Serviced Components" no .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwritingservicedcomponents.asp?frame=true>.

## **.NET Remoting**

O .NET Remoting fornece um mecanismo de chamado de procedimento remoto (RPC) flexível e extensível pelo qual componentes .NET podem se comunicar. O .NET Remoting permite utilizar uma grande variedade de protocolos de comunicação (como HTTP ou TCP), opções de codificação de dados (incluindo XML, SOAP e codificação binária) e vários modelos de ativação de objeto. Ele pode fornecer um meio de comunicação rápido e eficiente entre os objetos.

O .NET Remoting permite que você chame objetos remotos como se fossem objetos locais utilizando um objeto proxy que parece ser o objeto remoto. A infra-estrutura do .NET Remoting administra a interação entre o código do cliente e o objeto remoto através de chamadas de propriedade e método, codificando os dados que serão transmitidos entre eles e gerenciando a criação e exclusão do objeto remoto alvo.

A infra-estrutura do .NET Remoting exige que o cliente tenha conhecimento detalhado dos métodos e propriedades públicos do objeto remoto para oferecer um proxy do cliente. Uma maneira de garantir que o cliente tenha esse conhecimento é distribuir uma implementação completa do objeto remoto ao cliente. No entanto, é muito mais eficiente fatorar nos métodos e propriedades públicos para definições de interface e compilar essas interfaces em sua própria montagem. As montagens das interfaces podem ser utilizadas pelo cliente para oferecer um proxy mais adequado e eles podem ser utilizados pelo objeto remoto para implementar as funcionalidades necessárias. Essa técnica também permite que você atualize a implementação dos objetos remotos sem ter que redistribuir os objetos completos para o cliente.

É possível administrar a vida útil de objetos remotos de diversas maneiras. Objetos podem ser criados sob demanda para preencher uma única solicitação, ou você pode controlar sua vida útil mais precisamente utilizando um mecanismo de lease, onde o cliente mantém um lease no objeto remoto e este é mantido vivo pelo tempo que cliente queira utilizá-lo. O .NET Remoting também pode garantir que apenas uma instância de objeto exista para todos os clientes. Você pode escolher o tempo de vida de sua aplicação de acordo com suas necessidades de gerenciamento e escalabilidade de estado.

A infra-estrutura extensível do .NET Remoting permite que você crie canais personalizados e sinks. Os canais personalizados permitem que você defina a maneira na qual os dados serão transmitidos através da rede. Por exemplo, você pode definir um canal personalizado para implementar um protocolo wire personalizado. Sinks personalizados permitem que você intercepte e execute ações nos dados, conforme eles são enviados entre os objetos. Por exemplo, você pode definir um sink personalizado para criptografar ou comprimir os dados antes e depois da transmissão.

O .NET Remoting possui um poderoso e extensível mecanismo para comunicação entre objetos. No entanto, devido à sua natureza tightly coupled, ele pode não ser adequado para todas as situações. O .NET Remoting requer objetos implementados em .NET, tanto no cliente quanto no servidor, portanto, ele não é apropriado em situações nas quais a interoperabilidade entre dois ambientes diferentes é um

requisito. O .NET Remoting também não é adequado em situações nas quais a interação Tightly Coupled em estilo RPC entre cliente e servidor é inadequada. Por padrão, o .NET Remoting não fornece qualquer mecanismo built-in para criptografar ou para passar a identidade do usuário ou uma transação entre objetos. Nesse tipo de situação o Enterprise Services deve ser usado.

O .NET Remoting é uma boa opção, entretanto, para comunicação entre objetos em diferentes processos no computador cliente ou dentro dos limites de serviço ou para objetos em diferentes domínios de aplicação.

Para mais detalhes sobre utilização do .NET Remoting, consulte "An Introduction to Microsoft .NET Remoting Framework" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/introremoting.asp?frame=true>.

Para mais informações sobre a escolha entre Web Services e Remoting, consulte "ASP.NET Web Services or Remoting: How to Choose" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch16.asp?frame=true>.

## Message Queuing

Com o Microsoft Windows Message Queuing, fica fácil comunicar-se com as aplicações rápida e confiavelmente, enviando e recebendo mensagens. O Messaging oferece entrega de mensagens garantida e uma maneira confiável de efetuar muitos processos de negócios. O Message Queuing oferece um mecanismo de comunicação loosely coupled que pode ser utilizado dentro de sua aplicação Smart Client. O Message Queuing possui as seguintes características:

- **Entrega de mensagens garantida.** O Message Queuing garante que as mensagens serão entregues, apesar de falhas ou da ausência do sistema remoto, armazenando mensagens em uma fila até que possam ser entregues. Portanto, as mensagens são consideravelmente menos afetadas por falhas do que em chamadas diretas entre componentes.
- **Priorização de mensagens.** Mensagens mais urgentes ou mais importantes podem ser recebidas antes de mensagens menos importantes, ajudando a garantir tempo de resposta adequado para aplicações críticas.

**Observação:** Só é possível configurar prioridade de mensagens para mensagens não-transacionais.

- **Recursos Offline.** Se as mensagens não puderem ser entregues porque o cliente está offline, elas são armazenadas na fila de saída e entregues automaticamente quando o cliente ficar on-line novamente. Os usuários podem continuar a executar operações quando o acesso à fila de destino não estiver disponível. Enquanto isso, as operações adicionais podem prosseguir como se a mensagem já tivesse sido processada, porque a entrega de mensagens está garantida para quando a conexão de rede for restaurada.
- **Messaging transacional.** Você envia mensagens com parte de uma transação. Dessa forma, você pode enviar muitas mensagens relacionadas ou projetar sua aplicação para que ela participe de uma transação distribuída e garantir que todas as mensagens sejam entregues em ordem e apenas uma vez. Se qualquer erro ocorrer dentro de uma transação, toda a transação é cancelada e nenhuma mensagem é enviada.
- **Segurança.** A tecnologia do Message Queuing na qual o componente **MessageQueue** está baseado utiliza a segurança do Windows para proteger o controle de acesso, fornecer auditoria e criptografar e autenticar as mensagens que o seu componente envia e recebe. As mensagens do Message Queuing podem ser criptografadas sem conexão para torná-las impermeáveis a packet sniffers. Você também pode impedir que as filas recebam mensagens não criptografadas.

As aplicações que utilizam o Message Queuing podem enviar e ler mensagens a partir de filas utilizando as classes no espaço de nomes **System.Messaging**. A classe de **Message** condensa uma mensagem a ser enviada para uma fila, enquanto a classe **MessageQueue** oferece acesso a uma fila específica e a suas propriedades.

É necessário instalar e configurar o Message Queuing em todos os computadores que forem utilizá-lo. O Message Queuing está disponível para o sistema operacional Windows e para o Microsoft Windows CE .NET, permitindo que ele seja utilizado em dispositivos móveis como Pocket PCs.

O Message Queuing é uma boa opção de interação com serviços que ofereçam acesso baseado em mensagens. Você pode utilizá-lo para comunicar com outros sistemas que tenham o Message Queuing instalado. A interoperabilidade com outros sistemas é limitada, embora você possa utilizar toolkits de conectividade para se comunicar com outros sistemas de messaging como o MQSeries da IBM.

Para mais informações sobre como utilizar o Message Queuing, consulte "Message Queuing (MSMQ)" na documentação Microsoft Platform SDK em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msmq/msmq\\_overview\\_4ilh.asp?frame=true](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msmq/msmq_overview_4ilh.asp?frame=true).

Para informações sobre o MSMQ-MQSeries bridge programming, consulte "Programming Considerations Using MSMQ-MQSeries Bridge Extensions" em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/his/html/\\_sna\\_programming\\_considerations\\_using\\_msmq\\_mqseries\\_bridge\\_extensions\\_appl.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/his/html/_sna_programming_considerations_using_msmq_mqseries_bridge_extensions_appl.asp).

## Web Services

Um Serviço da Web é um componente de aplicação que:

- Expõe funcionalidades úteis a outras aplicações e Web Services através dos protocolos de Web Services padrão.
- Oferece descrição detalhada de suas interfaces, permitindo que você construa aplicações cliente que se comuniquem com elas. A descrição é oferecida em um documento XML chamado de documento Web Service Description Language (WSDL).
- Descreve suas mensagens utilizando um esquema XML.

As mensagens XML baseadas em SOAP dos Web Services podem ter partes explícitas (estruturada e escrita) ou partes Tightly Coupled (frouxamente definidas) usando arbitrariamente XML. Isso significa que os Web Services podem ser tanto loosely coupled ou tightly coupled e podem ser usados para implementar sistemas baseados em mensagens ou em estilo RPC, dependendo das exatas necessidades de seu ambiente.

Você pode usar para construir aplicações modulares dentro e através de organizações em ambientes heterogêneos. Essas aplicações são interoperáveis com uma ampla variedade de implementações, plataformas e dispositivos. Qualquer sistema que tenha capacidade de enviar XML através de HTTP pode utilizar serviços da Web. Como os Web Services são baseados em padrões, sistemas escritos em diferentes linguagens e para diferentes plataformas podem usar os serviços um dos outros. Isso é freqüentemente chamado de arquitetura orientada a serviços.

Os principais padrões utilizados nos Web Services são HTTP, SOAP, UDDI, e WSDL. Web Services independem do transporte de protocolos. No entanto, HTTP é o mecanismo mais comum para transporte de mensagens SOAP. Portanto, Web Services são adequados para aplicações que passam por redes e firewalls corporativos, como Smart Clients que precisam comunicar-se com os serviços da internet.

Diversos padrões de Web Services estão surgindo para ampliar a funcionalidade dos Serviços da Web. O Microsoft Web Service Enhancements (WSE) 2.0 suporta padrões Web Services recentes, como WS-Security, WS-SecureConversation, WS-Trust, WS-Policy, WS-Addressing, WS-Referrals e WS-Attachments e Direct Internet Message Encapsulation (DIME). O WSE oferece um modelo de programação para implementar as diversas especificações que ele suporta.

Para mais informações consulte "Web Service Enhancements (WSE)" em <http://msdn.microsoft.com/webservices/building/wse/default.aspx>.

Para mais informações sobre o SOAP consulte "Understanding SOAP" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/understandsoap.asp>.

Para mais informações sobre o WS-Security consulte "Web Services Security Specifications Index Page" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wssecindex.asp>.

As comunicações dos Web Services podem ser coarse-grained, self-contained e stateless. No entanto, Web Services são geralmente bastante eloquentes se comparados a outras formas de comunicação.

Os Web Services são a melhor abordagem para a construção da maioria das aplicações Smart Client. O alto grau de interoperabilidade permite que os Web Services se comuniquem com um grande número de aplicações. O uso de padrões largamente adotados significa que eles podem passar através da infra-estrutura de rede e de firewalls com configuração adicional mínima (comparado a outras tecnologias que precisam que portas proprietárias sejam abertas). O suporte a Web Services no sistema de desenvolvimento Microsoft Visual Studio® significa que você pode trabalhar com eles em um único ambiente de desenvolvimento.

Os Web Services podem não ser adequados em aplicações de alto desempenho porque são eloquentes e possuem payloads de mensagens relativamente pesados, se comparado a outras tecnologias de mensagens como o .NET Remoting e o Message Queuing.

Para mais informações sobre como construir e utilizar Web Services consulte "XML Web Services Created Using ASP.NET and XML Web Service Clients" no .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconaspnetbuildingwebservicesaspnetwebseviceclients.asp?frame=true>.

## Escolhendo uma Opção de Comunicação

Opções de comunicação diferentes são adequadas em situações diferentes. A Tabela 3.1 resume as diferentes opções para conectar-se.

**Tabela 3.1: Opções Smart Client**

Opção	Vantagens	Desvantagens
Enterprise Services	Provê acessos aos serviços COM+ Permite que a identidade flua com a chamada	Requer Serviced Components instalados no cliente Utilizável somente para o mesmo processo ou computador
.NET remoting	Rápido Ajustável Suporta protocolos customizados	Requer o .NET Framework para executar Proprietário Não atravessa Firewall sem a abertura de portas RPC Não há Infra-estrutura de segurança

(continua)

Opção	Vantagens	Desvantagens
Message Queuing	Útil para comunicação com sistemas de mensagens Seguro Garante a entrega da mensagem	Requer que o Message Queuing seja configurado no cliente A integração com outros sistemas não é fácil
Web services	Suporta integração Extensível Suporte forte da indústria Padrões claramente definidos Agnóstico quanto a fornecedores / linguagens Seguro	Carregado de informações descritivas Desempenho mais lento que .NET remoting

Como ilustra a Tabela 3.1, há algumas situações nas quais o Enterprise Services, NET Remoting e o Message Queuing podem ser tecnologias apropriadas para comunicação entre Smart Clients e os recursos conectados. No entanto, na maioria dos casos, os Web Services são os melhores mecanismos para conectar aplicações Smart Client aos serviços.

Uma arquitetura construída com base na comunicações de Web Services pode funcionar bem, tanto em ambientes conectados como offline. A dependência de protocolos de Internet permite uma larga distribuição do cliente a qualquer um na Internet.

## Projetando Aplicações Smart Client Conectadas

Durante o projeto de seu Smart Client, há algumas recomendações que se considerar, incluindo:

- **Utilizar mensagens coarse-grained, em cápsula.**
- **Evitar transações ACID distribuídas.**
- **Evitar o envio de datasets através da rede.**
- **Dividir grandes datasets.**
- **Versione seus assemblies e Web Services.**

### Utilizar Mensagens Coarse-Grained, em cápsula

Chamadas de rede distribuídas são operações caras. Você não deveria projetar a sua interface externa da mesma maneira *fine-grained* que você projetaria interfaces locais ou o desempenho será afetado. Para evitar dependências entre as mensagens, é uma boa idéia construir métodos de interface como funções self-contained. Isso evita escrever complexos códigos de reconciliação de rastreamento para manipular a falha de uma mensagem que dependa do êxito da conclusão de outra.

### Evitar Transações ACID Distribuídas

Transações ACID (Atômicas, Consistentes, Isoladas, Duráveis) distribuídas são resource-intensive com muito tráfego de rede e muitos sistemas de bloqueio interdependentes em transações locais pendentes. Se o seu Smart Client ou serviço estiver aguardando uma resposta e não pode continuar até que essa resposta seja recebida, uma transação ACID distribuída pode bloquear os processos de negócios.

Os problemas de transações ACID distribuídas serão exacerbados se houver probabilidade de seus Smart Clients mudarem para o modo offline sem aviso. Nesse caso, um cliente pode colocar uma trava nos dados e ficar offline antes que essa trava possa ser liberada no servidor.

Se não puder evitar dependência de mensagens dividindo as interfaces em mensagens discretas únicas, você terá diversas opções de como lidar com as transações e ainda evitar transações ACID distribuídas:

- Envie mensagens tightly coupled ao servidor e utilize um coordenador de transações como o Microsoft BizTalk® Server para manipular as dependências de mensagens.
- Escreva você mesmo códigos de compensação de transações no cliente ou servidor. Utilize um protocolo de comunicações que o servidor possa usar para decidir quando iniciar uma transação e como notificar o cliente com relação ao êxito ou falha da conclusão da transação processada em sua totalidade.

## Evitar o Envio de Datasets Através da Rede

Datasets podem ser grandes e densos demais para serem usados como mecanismos de carga para envio de dados através de múltiplas camadas. Em vez disso, você deveria usar objetos de transferência de dados (DTOs) para diminuir a carga de mensagens em suas interfaces externas. Para mudança de dados, você deveria considerar o envio somente dos dados modificados em vez de todos os dados. Para mais informações sobre DTOs consulte o Capítulo 2, "Manipulando Dados."

## Dividir Grandes Datasets

Grandes datasets podem causar problemas de desempenho no cliente se você tentar apresentá-los ao mesmo tempo. Portanto, é necessário dividi-los em datasets menores. Essa divisão de dados é conhecida como *paging*. Por exemplo, em vez de mostrar todo o conteúdo de uma lista telefônica, você pode escolher mostrar uma página por vez (por exemplo, 20 registros por tela, dispostos alfabeticamente). Se você projetar o cliente para que ele use o *paging*, deve assegurar que a interface de usuário esteja projetada para facilitar a navegação entre páginas para o usuário.

Esse conceito de dividir grandes datasets também se aplica à comunicação com o servidor através da rede. Se dividir os dados em pedaços gerenciáveis, então poderá carregar os dados solicitados conforme forem necessários, técnica conhecida como *lazy loading*. No exemplo da lista telefônica, apenas os dados necessários para a operação atual seriam carregados, reduzindo o impacto na aplicação e na rede e potencialmente tornando ambas mais responsivas.

Para melhorar a experiência do usuário, utilize encadeamentos adicionais para executar processamento e comunicação em segundo plano com serviços, antecipando solicitações dos usuários.

Embora o suporte para lazy loading possa ser um importante aspecto do projeto de sua aplicação Smart Client, você deve ter em mente as necessidades offline de sua aplicação. Lazy loading de dados que trafegam através da rede pode impedir que sua aplicação funcione offline conforme desejado.

## Versione seus Assemblies e Web Services

Quando você faz atualizações e lança novas versões do seu software Smart Client para os clientes, crie novas versões das montagens. Se você utiliza montagens versionadas e projeta seus Web Service para suportar interfaces retrocompatíveis, você pode suportar versões múltiplas do software do cliente. Ao lançar novas versões dos seus serviços da Web, diferencie-os com uma convenção de nome canônica. Mudar os espaços de nome de cada lançamento para que eles contenham informações de data de modo a tornar clara a versão dos Web Services com a qual o cliente está se comunicando.

Para mais detalhes sobre como lidar com múltiplas versões de montagens, consulte o Capítulo 7, "Implantando e Atualizando Aplicações Smart Client."

## Sumário

Os Smart Clients precisam acessar recursos locais ou remotos para funcionar. A maneira como você lida com essas comunicações é um fator crítico para seu sucesso ao projetar Smart Clients confiáveis e responsivos ao usuário. Requisitos como desempenho, segurança e flexibilidade afetam a escolha de conectividade apropriada para o seu ambiente. Com as orientações deste capítulo, determine que formas de conectividade são ideais para seus Smart Clients e projetar adequadamente os Smart Clients e os recursos com os quais se comunicam.



# 4

## Smart Clients Ocasionalmente Conectados

Vivemos em um mundo cada vez mais conectado. No entanto, não podemos confiar na conectividade todo o tempo. Seus usuários podem viajar, perder sua conexão sem fio temporariamente, ter problemas de latência ou de largura de banda, ou você pode precisar desligar partes de sua rede para manutenção. Mesmo que os usuários tenham uma boa conexão com a rede, as aplicações podem não ser capazes de acessar recursos de rede todo o tempo. Um serviço solicitado pode estar ocupado, sem acesso ou apenas indisponível temporariamente.

Uma aplicação é chamada de *ocasionalmente* conectada se em alguns momentos ela fica impedida de interagir com os serviços ou dados através de uma rede. Se você puder permitir que seus usuários sejam produtivos em suas aplicações quando offline e ainda oferecer-lhes os benefícios de uma aplicação conectada quando a conexão estiver funcionando, aumentará a produtividade e eficiência do usuário e a aproveitamento de suas aplicações.

Um dos benefícios principais dos Smart Clients sobre as aplicações baseadas na Web é que eles permitem que os usuários continuem trabalhando quando a aplicação não consegue conectar-se aos recursos da rede. Smart Clients ocasionalmente conectados são capazes de executar seu trabalho quando não estiverem conectados aos recursos da rede e atualizar recursos da rede em segundo plano posteriormente. A atualização ocorrem quase que imediatamente, mas em alguns momentos, podem ocorrer dias ou até semanas depois.

Para conceder a uma aplicação ocasionalmente conectada plenos recursos, é necessário oferecer uma infra-estrutura que permita aos usuários trabalhar mesmo quando não tiverem uma conexão aos recursos de rede. Essa infra-estrutura inclui cache de dados, de forma que todos os dados solicitados estejam disponíveis no cliente e armazenamento de detalhes do trabalho do usuário, que serão utilizados para sincronizar o cliente e os recursos de rede quando o usuário voltar a ficar on-line. Os exatos recursos e capacidades que sua aplicação precisa para suportar operações ocasionalmente conectadas dependem da sua conectividade, do ambiente operacional e da funcionalidade que o usuário espera ter ao estar on-line e offline. Entretanto, todas as aplicações Smart Client devem oferecer algum tipo de experiência aos usuários quando não estiverem conectadas à rede, mesmo que extremamente limitada. Ao projetar e construir suas aplicações, evite sempre gerar mensagens de erro no cliente pelo fato de um servidor não estar disponível.

Esse capítulo examina os problemas que você pode encontrar ao construir aplicações com capacidades offline. Ele revisa estratégias diferentes para projetar aplicações offline, discute considerações de design em detalhes, examina como estruturar aplicações para usar tarefas e examina como suas aplicações devem manipular dados.

## Cenários comuns ocasionalmente conectados

Smart clients ocasionalmente conectados são extremamente úteis em muitas situações comuns. Muitos cenários offline envolvem a necessidade de o usuário desconectar-se da rede e trabalhar sem uma conexão de rede, como por exemplo:

- Um agente de seguros pode precisar criar uma nova apólice de seguro enquanto está fora do escritório. Ele digita todos os dados relevantes, calcula prêmios e emitir detalhes de apólice sem a possibilidade de conectar-se ao sistema do escritório.
- Um representante de vendas pode ter de fazer um grande pedido ainda conversando com o cliente, em algum lugar onde o representante não possa se conectar ao servidor. Ele consulta as listas de preços e informações de catálogo, digita todos os dados do pedido e oferece estimativas de entrega e níveis de desconto sem se conectar.
- Um técnico de manutenção pode precisar de informações técnicas detalhadas ao atender um chamado de um cliente. A aplicação ajuda-o a diagnosticar o problema, oferecer documentação e detalhes técnicos e permite que o técnico faça um pedido de peças e documente todas as ações feitas sem ter de se conectar.

Outros cenários offline envolvem uma conectividade intermitente ou de baixa qualidade, como por exemplo:

- A conectividade entre centros de atendimento ao consumidor por todo o mundo e a rede da corporação central pode não ser de alta qualidade suficiente para permitir o uso on-line a todo o momento. A aplicação deve oferecer capacidades offline, incluindo cache de dados, de modo que a aproveitamento da aplicação seja mantida.
- Juntas médicas viajando com Tablet PCs podem passar por interrupções na conectividade de rede enquanto viajam. Quando a aplicação é conectada, ela deve sincronizar os dados em segundo plano e não devem esperar por uma reconexão explícita.

Smart clients ocasionalmente conectados devem ser desenvolvidos de modo a tirar máxima vantagem de uma conexão quando ela estiver disponível, garantindo que aplicações e dados estejam o mais atualizados possível, sem afetar o desempenho da aplicação.

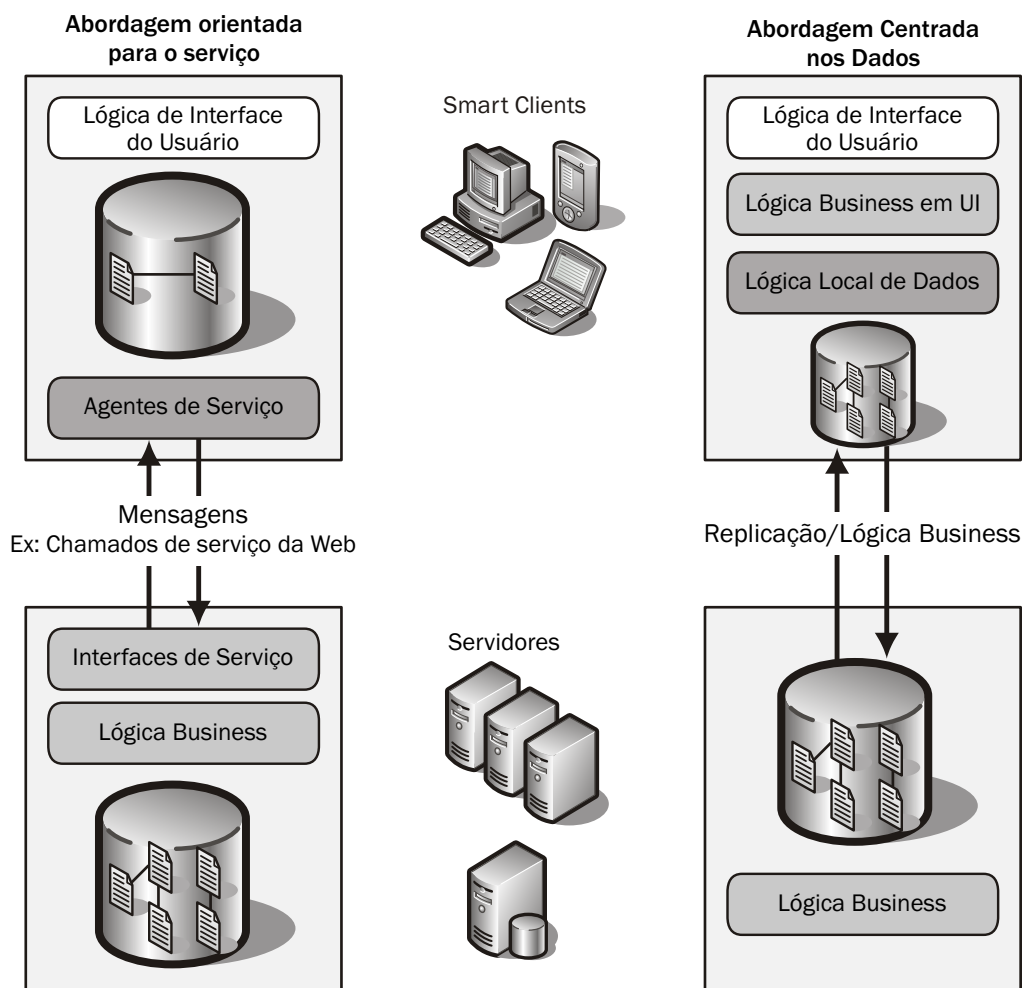
## Estratégias de Projeto de Ocasionalmente Conectados

Há duas abordagens gerais para a arquitetura de Smart Clients ocasionalmente conectados: *centrados nos dados* e *orientados para o serviço*.

As aplicações que trabalham na estratégia centrada nos dados têm um sistema de gerenciamento de banco de dados relacional (RDBMS) instalado localmente no cliente e utilizam os recursos integrados do sistema de banco de dados para propagar alterações de dados locais de volta ao servidor, manipular o processo de sincronização e detectar e solucionar quaisquer conflitos de dados.

As aplicações que trabalham na abordagem orientada para o serviço armazenam informações em mensagens e as organizam em filas quando o cliente está offline. Após a conexão ser restabelecida, as mensagens em fila são enviadas ao servidor para processamento.

**Figura 4.1** *exibe as abordagens centradas nos dados e orientadas para o serviço*



**Figura 4.1**

*Abordagem orientada para o serviço x centrada nos dados para desenvolvimento de aplicações ocasionalmente conectadas*

Essa seção examina as duas abordagens em maiores detalhes e explica quando você deve utilizar cada uma.

## Abordagem Data-Centric

Quando você utiliza a abordagem data-centric, o servidor geralmente publica os dados e o cliente cria uma assinatura para os dados que ele precisa, de forma que possa copiar esses dados para o armazenamento local antes de o cliente ficar offline. Quando o cliente está offline, ele faz alterações nos dados locais por meio de chamadas ao armazenamento local de dados. Quando o cliente volta a ficar on-line, o armazenamento de dados propaga as alterações feitas aos dados no cliente de volta ao servidor. As alterações feitas aos dados no servidor também são propagadas de volta para o cliente. Quaisquer conflitos encontrados durante a fase de mesclagem são manipulados de acordo com as regras de solução de conflitos especificadas no servidor ou no cliente, de acordo com regras personalizadas definidas pelo analista de negócios.

O processo de mesclagem de alterações entre o cliente e o servidor é conhecido como *merge replication*. As alterações ocorrem de maneira autônoma tanto no cliente quanto no servidor, assim, as transações ACID não são usadas. Em vez disso, quando uma mesclagem é executada, todos os assinantes do sistema utilizam os valores de dados fornecidos pelo publisher (Publicador).

A principal vantagem da abordagem centrada nos dados é que todo código de monitoramento de alteração fica dentro do banco de dados relacional. De maneira geral, isso inclui o código para detecção de conflito nos níveis de linha e coluna do banco de dados, código de validação de dados e restrições. Isso significa que você não precisa escrever seu próprio código de monitoramento de alteração ou detecção e solução de conflito, ainda que precise estar ciente do esquema de *merge-replication* de modo a otimizar suas aplicações para conflitos e atualizações de dados.

No modelo centrado nos dados, o sistema de banco de dados manipula a sincronização; portanto, você não precisa implementar toda a funcionalidade de sincronização de dados sozinho. Os usuários definem quais tabelas precisam de sincronização de dados, e o sistema de banco de dados permite que a infra-estrutura monitore as alterações e detecte e solucione conflitos. Você estende a infra-estrutura a fim de solucionar ou evitar conflitos personalizados por meio de solucionadores personalizados que utilizam objetos COM ou procedimentos armazenados Transact SQL (TSQL). Além disso, por haver um único repositório de dados no sistema, a convergência de dados é garantida entre um servidor e um cliente ao término da sincronização.

Há, contudo, algumas desvantagens na abordagem centrada nos dados. A necessidade de um banco de dados local no cliente significa que a abordagem pode não ser apropriada nas seguintes situações:

- Se a aplicação for executada em um pequeno equipamento
- Se o uso de um mecanismo *light-touch* for requerido
- Se usuários não administradores precisarem implementar a aplicação

A Microsoft oferece software de banco de dados que pode ser executado em plataformas Windows® client, Windows Server™ Pocket PC, mas não fornece software de banco de dados para equipamentos SmartPhone.

Além disso, o *tight coupling* entre o banco de dados do servidor e o do cliente significa que as alterações feitas ao esquema de banco de dados no servidor terão um impacto direto no cliente. Isso dificulta o gerenciamento de alterações em esquemas de banco de dados no cliente ou servidor.

Com um grande número de clientes, há uma necessidade de oferecer uma forma gerenciável e escalável de implementar conjuntos diferentes de dados. A *merge replication* suporta filtro dinâmico, o que permite que o administrador defina esses conjuntos de dados offline e implemente-os de uma forma escalável. Você deve aproveitar do mecanismo de filtro oferecido pelo banco de dados para reduzir a quantidade de dados a ser enviada entre o cliente e o servidor e reduzir a probabilidade de conflitos.

Há muitos benefícios usando um banco de dados local para armazenar e manipular dados localmente. Um deles é usar o banco de dados para propagar alterações locais de volta ao servidor e ajudar a resolver problemas de sincronização. Você deve usar a abordagem centrada nos dados quando:

- Implementar uma instância de banco de dados no cliente.
- Sua aplicação funcionar em um ambiente de duas camadas.
- Acoplar firmemente o cliente ao servidor através de definições de esquemas de dados e protocolos de comunicação.
- Quiser monitoramento e sincronização de alteração integrados.

- Quiser confiar no banco de dados para manipular conflitos de reconciliação de dados e minimizar a quantidade de códigos personalizados de reconciliação que precisam ser escritos.
- Não necessitar interagir com múltiplos serviços distintos.
- Os usuários do Windows se conectarem a um banco de dados diretamente através de uma rede local (LAN) ou de uma rede particular virtual (VPN/IPSec). As aplicações escritas para a plataforma Pocket PC podem sincronizar HTTP por meio de HTTPS.

**Observação:** Esse guia não cobre a abordagem centrada nos dados em profundidade. Ela é encontrada com mais detalhes em muitos lugares, incluindo o Microsoft SQL Server Books On-line ou MSDN. Para maiores detalhes sobre a abordagem centrada nos dados, consulte "Merge Replication" em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/replsql/repltypes\\_6my6.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/replsql/repltypes_6my6.asp).

## Abordagem Orientada para o Serviço

Com a abordagem orientada para o serviço, o cliente interage com quaisquer serviços solicitados. Além disso, o cliente está focado nas próprias solicitações de serviço, em vez de proceder às alterações diretas a dados mantidos localmente. As solicitações de serviço levam a alterações de estado no cliente ou servidor, mas tais alterações são sub-produtos das solicitações de serviço.

Uma vantagem da estratégia orientada para o serviço é que não é necessário o uso de um banco de dados relacional local no cliente. Isso significa que a abordagem é aplicada a muitos tipos diferentes de clientes, incluindo aqueles com uma quantidade pequena de poder de processamento, como telefones celulares.

Uma abordagem orientada para o serviço é particularmente apropriada quando sua aplicação opera em um ambiente de Internet e extranet. Se seu cliente opera fora do firewall e interage com serviços corporativos, ao usar a estratégia orientada para o serviço, você evita a necessidade de abrir portas específicas no firewall como, por exemplo, para habilitar o acesso direto a um banco de dados ou ao Microsoft Message Queuing (MSMQ).

O *loose coupling* significa que você utilizar diferentes esquemas de dados no cliente e transformar os dados no cliente. Na verdade, o cliente e o servidor não precisam ter conhecimento um do outro. Você também atualiza os componentes dos dois de maneira independente.

A principal desvantagem dessa abordagem é a escrita de mais códigos de infra-estrutura para facilitar o armazenamento e encaminhamento de mensagens, bem como a detecção de quando a aplicação estiver on-line ou offline. Isso oferece mais flexibilidade no projeto, mas geralmente significa mais trabalho para criar seus clientes offline.

---

**Nota:** O Smart Client Offline Application Block oferece um código que suporta uma estratégia orientada para o serviço para clientes offline. Você pode usar esse bloco para detectar quando uma aplicação está on ou offline e armazenar e encaminhar mensagens para processamento em um servidor. Para uma visão geral sobre esse bloco de aplicação, consulte Smart Client Offline Application Block em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/offline.asp>.

---

A abordagem orientada para o serviço é mais adequada para clientes que necessitam interagir com diversos serviços diferentes. Pelo fato da carga da mensagem estar encapsulada, a camada de transporte varia sem afetar o conteúdo das mensagens. Por exemplo, uma mensagem destinada originalmente para um serviço da Web poderia ser enviada com a mesma facilidade para um serviço que consumisse mensagens Message Queuing. O fato de a mensagem ser transport agnostic também permite implementações personalizadas de segurança pela aplicação.

Você deve usar a abordagem orientada para o serviço quando:

- Desejar separar o cliente e o servidor a fim de permitir versões e usos independentes.
- Requerer maior controle e flexibilidade sobre problemas de reconciliação de dados.
- Possuir experiência no desenvolvimento para escrever códigos mais avançados de infra-estrutura de aplicação.

- Requerer um cliente enxuto.
- For capaz de estruturar sua aplicação em uma arquitetura orientada para o serviço.
- Requerer funcionalidade de negócios específica (por exemplo, regras e processamento de negócios personalizados, reconciliação flexível, etc).
- Precisar ter controle sobre esquemas de dados armazenados no cliente e flexibilidade que possa ser diferente do servidor.
- Sua aplicação interagir com múltiplos ou diferentes serviços (por exemplo, Web Services múltiplos ou serviços através de mecanismos Message Queuing, Web Services ou RPC).
- Precisar de um esquema de segurança personalizado.
- Sua aplicação operar em um ambiente de Internet ou extranet.

Enquanto ambas abordagens (centradas nos dados e orientada para o serviço) são abordagens arquiteturais válidas, muitas aplicações Smart Client não são capazes de suportar instâncias relacionais de banco de dados completas no cliente. Nesses casos, adotar uma abordagem orientada para o serviço e assegurar que tenha a infra-estrutura apropriada para manipular questões como cache de dados e detecção e solução de conflitos.

Por essa razão, o restante desse capítulo está focado nas questões que desenvolvedores Smart Client precisam levar em consideração ao implementar a abordagem orientada para o serviço.

## Projetando Aplicações Smart Client Ocasionalmente Conectadas Utilizando uma Abordagem Orientada para o Serviço

Ao desenvolver seus Smart Clients ocasionalmente conectados utilizando uma abordagem orientada para o serviço, há uma série de questões que precisam ser levadas em consideração, incluindo:

- **Favorecimento de comunicação assíncronica.**
- **Minimização de interações complexas de rede.**
- **Adição de recursos de cache de dados.**
- **Gerenciamento de conexões.**
- **Desenvolvimento de um mecanismo Store-and-Forward.**
- **Gerenciamento de dados e de conflitos de regras de negócios.**
- **Interação com Web Services CRUD (criação, leitura, atualização e exclusão).**
- **Uso de uma abordagem baseada em tarefa.**
- **Manipulação de dependências.**

Essa sessão discute essas questões com maiores detalhes:

### Favorecimento de Comunicação Assíncronica

As aplicações utilizam um dos dois métodos de comunicação ao interagir com dados e serviços localizados na rede:

- **Comunicação síncronica.** A aplicação é desenvolvida de modo a esperar uma resposta antes de continuar o processamento (por exemplo, comunicação RPC síncronica).
- **Comunicação assíncronica.** A aplicação comunica-se utilizando uma mensagem ou algum outro tipo de transporte baseado em mensagem e tem um atraso entre a solicitação e qualquer resposta ou não espera qualquer resposta.

---

**Nota:** Nesse guia, a comunicação síncrona refere-se a toda comunicação que espera uma resposta antes que o processamento possa continuar, mesmo que o chamado síncrono seja executado em um encadeamento em uma base separada.

---

Se você estiver desenvolvendo uma nova aplicação Smart Client, certifique que ela use primeiramente uma comunicação assíncrona ao interagir com dados e serviços localizados na rede. Aplicações criadas para esperar um atraso entre a solicitação e a resposta são mais apropriadas para o uso ocasionalmente conectado, contanto que a aplicação forneça uma funcionalidade significativa e útil enquanto o usuário espera por uma resposta, além de não impedir que ele continue seu trabalho caso a resposta demore.

Quando a aplicação não estiver conectada aos recursos da rede, armazene as solicitações localmente e envie-as ao serviço remoto quando a aplicação se conectar novamente. Tanto nos casos on-line e offline, pelo fato de a aplicação não estar esperando uma resposta imediata para uma solicitação, o usuário não é impedido de continuar a usar a aplicação e continuar trabalhando.

As aplicações que utilizam a comunicação síncrona, mesmo em um encadeamento de background (base), não são adequadas para serem ocasionalmente conectadas. Assim, você minimiza o uso de comunicações síncronas em seus Smart Clients. Se você estiver projetando novamente uma aplicação que utilize comunicação síncrona para ser um Smart Client, assegure-se que ela adote um modelo mais assíncrono de comunicação, de modo a que possa funcionar offline. Entretanto, em muitos casos, você pode implementar uma comunicação do tipo síncrona por sobre de uma infraestrutura assíncrona (conhecida como modelo sync-on-async) de modo que sejam feitas poucas alterações ao design da aplicação.

Permitir que suas aplicações utilizem a comunicação assíncrona pode trazer benefícios que vão além do uso ocasionalmente conectado. A maior parte das aplicações criadas para comunicação assíncrona são mais flexíveis do que as que utilizam comunicações síncronas. Por exemplo, uma aplicação assíncrona pode se desligar em meio a uma tarefa sem afetar o processamento de solicitações e respostas quando ela for iniciada novamente.

Na maioria dos casos, você não precisa implementar os dois tipos de comportamento (síncrono e assíncrono) em uma aplicação para uso on-line e offline. Um comportamento assíncrono é apropriado para os dois tipos de uso, as solicitações são processadas praticamente em tempo real quando a aplicação estiver on-line.

## **Minimização de Interações Complexas de Rede**

Smart clients ocasionalmente conectados devem minimizar ou eliminar interações complexas com dados e serviços localizados em rede. Quando sua aplicação estiver offline, ela armazena as solicitações e envia-as quando a aplicação se conectar novamente, ou ela pode ter que esperar pelas respostas. Em qualquer um dos casos, a aplicação não é capaz de saber imediatamente se uma solicitação irá obter sucesso ou se já o obteve.

Para permitir que sua aplicação continue trabalhando offline, faça algumas suposições sobre o sucesso das solicitações de rede ou alterações aos dados locais. Monitorar essas suposições e dependências entre solicitações de serviço e alterações de dados pode ser complexo. Para facilitar essa tarefa, simplifique as interações de rede de suas aplicações Smart Client o máximo possível.

Geralmente, as solicitações que não devolvem nenhum dado (solicitações dispare-e-esqueça, ou fire-and-forget) não são um problema para aplicações ocasionalmente conectadas; a aplicação pode armazenar a solicitação e encaminhá-la quando ela se conectar novamente. Quando a aplicação estiver offline, ela não sabe se houve êxito na chamada; portanto, deve supor esse êxito na chamada. Isso pode influenciar todo o processamento posterior.

Se uma solicitação retornar dados requeridos antes de a aplicação continuar trabalhando, sua aplicação deve usar valores provisórios ou dummy, ou funcionar sem os dados. Nesse caso, desenvolva a aplicação para que ela monitore dados confirmados e provisórios, e que a interface do usuário informe o usuário de dados que sejam provisórios ou estejam pendentes. Isso permite ao usuário ou à aplicação tomar decisões informadas baseadas na validade dos dados, além de evitar problemas de conflitos e corrupção de dados posteriormente.

Nas situações nas quais o usuário conclui um número discreto de unidades de trabalho offline, permita na sua aplicação que cada unidade de trabalho falhe ou tenha sucesso por si. Por exemplo, em uma aplicação onde o usuário digita informações de pedido, a aplicação permite que o usuário digite tantos pedidos quantos forem necessários, mas deve garantir que um pedido não dependa do sucesso de outro.

É relativamente fácil garantir que não haja dependências entre unidades de trabalho quando a aplicação faz apenas uma solicitação de serviço por unidade de trabalho. Isso permite que sua aplicação monitore solicitações pendentes e as processe quando ficar on-line. Contudo, em algumas situações, as tarefas do usuário são mais complicadas e múltiplas solicitações de serviço têm de ser feitas para completar essas tarefas. Nesses casos, a aplicação certifica que cada solicitação seja consistente com as outras de modo a manter a integridade dos dados.

## **Adição de Recursos de Cache de Dados**

Sua aplicação precisa garantir que todos os dados necessários para o usuário continuar trabalhando estejam disponíveis no cliente quando ele ficar offline. Em alguns casos é necessário armazenar dados no cliente para propósitos de desempenho, ainda que em muitas vezes sua aplicação deva armazenar dados adicionais para permitir o uso ocasionalmente conectado. Por exemplo, os dados voláteis não são armazenados para uma aplicação desenvolvida para ser usada on-line, mas habilitam a mesma para que ela trabalhe offline requerendo que os dados sejam armazenados no computador local. Tanto o cliente quanto o servidor devem ser desenvolvidos tendo em vista a volatilidade dos dados, de modo que eles manipulam as atualizações e conflitos de maneira apropriada.

Quando uma aplicação estiver offline, você pode optar por não excluir dados desatualizados do cache de dados da aplicação e permitir, por sua vez, que o usuário continue trabalhando com os dados desatualizados. Em outros casos, a aplicação os exclui automaticamente do cache para evitar que os usuários utilize-os, causando problemas futuros. No último caso, a aplicação pode parar de fornecer a funcionalidade requerida até que os novos dados tenham sido obtidos através de um processo de sincronização.

A atualização de dados no cache ocorrem de diversas maneiras, dependendo do estilo e da funcionalidade da sua aplicação. Para algumas, os dados armazenados em cache são atualizados automaticamente ao terminar sua validade, de acordo com algum tipo de planejamento, quando a aplicação executar uma operação síncrona ou quando o servidor alterar os dados e informar a aplicação sobre a mudança. Outras aplicações permitem ao usuário selecionar os dados manualmente para serem armazenados, permitindo ao mesmo examinar ou trabalhar com os dados enquanto estiver offline.

Outras considerações sobre armazenamento de dados também se aplicam, como segurança e restrições de manipulação de dados. Essas questões não são encontradas somente em aplicações com capacidade offline e são descritas com maior profundidade no Capítulo 2, "Manipulação de Dados".

## **Manipulação de Alterações em Dados de Referência**

Dados de referência mudam com pouca frequência. Geralmente, as aplicações incluem uma quantidade significativa desse tipo de dados. Por exemplo, no registro de um consumidor, o nome de um



consumidor não muda com frequência. Esses tipos de dados são armazenados facilmente em cache no cliente, mas, às vezes, seus dados de referência irão mudar e você terá que possuir um mecanismo para propagar as alterações para seus Smart Clients.

Você tem duas opções para propagar os dados: o modelo push e o modelo pull.

No modelo push, o servidor notifica o cliente de forma pró-ativa e tenta empurrar (push) os dados para fora. Na abordagem centrada nos dados, isso poderia ser uma mensagem contendo os dados atualizados. (Isso requer, do cliente, a implementação de um ponto final ao qual o servidor possa se conectar).

No modelo pull, o cliente contata o servidor para uma atualização. O cliente pode fazer isso verificando regularmente o servidor ou examinando os metadados com os dados originais que dizem quando a validade dos dados de referência termina. O cliente pode, ainda, puxar dados do servidor (uma lista de preços, por exemplo) e usá-los apenas quando se tornarem válidos.

Em alguns casos, você pode optar por adotar um modelo no qual o servidor notifique o cliente quando houver uma atualização disponível (por exemplo, enviando um alerta quando o cliente se conectar), e o cliente, por sua vez, puxará os dados do servidor.

## Gerenciamento de Conexões

Ao desenvolver seus Smart Clients ocasionalmente conectados, considere o ambiente no qual sua aplicação opera, tanto em termos de conectividade disponível, quanto em termos de comportamento desejado de sua aplicação conforme alterações nessa conectividade.

Algumas aplicações são desenvolvidas para operar por longos períodos de tempo (dias ou mesmo semanas) sem uma conexão. Outras desenvolvidas para esperar uma conexão em todas as vezes, mas que tenham a habilidade de lidar com uma desconexão temporária sem maiores problemas. Algumas aplicações oferecem apenas um subconjunto de funcionalidades quando em modo offline, ao passo que outras a maioria das funcionalidades são para uso offline.

Enquanto muitos cenários ocasionalmente conectados envolvem a necessidade de o usuário desconectar-se explicitamente da rede e trabalhar sem uma conexão, às vezes a aplicação está offline sem que ela esteja realmente desconectada da rede. Suas aplicações podem ser desenvolvidas para lidar com um ou ambos os tipos de cenários.

### Gerenciamento de Conexão Manual

Sua aplicação pode ser desenvolvida para funcionar quando o usuário decidir trabalhar offline. A aplicação deve armazenar todos os dados que o usuário precisa no computador local. Nesse caso, o usuário interage com a aplicação sabendo que ela está offline, e a aplicação não tenta executar operações em rede até que ela receba o comando de entrar no modo on-line e executar uma operação de sincronização.

Você também pode incluir o suporte para usuários para notificar a aplicação quando eles estiverem usando uma conexão que seja de alto custo de conexão ou de largura estreita, como um hotspot comercial sem fio ou uma conexão discada. Nesse caso, a aplicação é desenvolvida de modo a unir solicitações para que quando uma conexão for estabelecida o uso possa ser maximizado.

### Gerenciamento Automático de Conexão

Sua aplicação pode ser desenvolvida para adaptar-se dinamicamente quando houver alterações inesperadas na conectividade. Essas alterações podem incluir:

- **Conectividade intermitente.** A aplicação adapta-se a lidar sem problemas com as ocasiões nas quais a conexão de rede é temporariamente perdida. Algumas suspendem temporariamente a

funcionalidade até que a mesma possa voltar ao modo on-line , enquanto outras aplicações oferecem funcionalidade total.

- **Qualidade variável de conexão.** A aplicação antecipa o fato de que a conexão de rede tenha largura estreita ou alta latência ou determina isso de maneira dinâmica e alterar o comportamento de modo a adequar-se ao ambiente. Se a qualidade de conexão piorar, a aplicação armazena dados em cache de maneira mais agressiva.
- **Disponibilidade variável de serviço.** A aplicação lida com a indisponibilidade de serviços com os quais ela normalmente interage e alternar para o comportamento offline. Se a aplicação interagir com mais de um serviço e um deles tornar-se indisponível, ela considera todos os serviços offline.

Você pode detectar se uma aplicação Smart Client tem conectividade usando a wininet.dll. Trata-se da mesma DLL utilizada pelo Microsoft Internet Explorer para determinar se os usuários estão conectados à Internet. O seguinte código ilustra como chamar a wininet.dll.

```
[DllImport("wininet.dll")]
private extern static bool InternetGetConnectedState( out int
connectionDescription, int reservedValue ) ;
public bool IsConnected() {
int connectionDescription = 0;
return InternetGetConnectedState(out connectionDescription, 0);
}
```

## Desenvolvimento de Mecanismo Store-and-Forward

Se desenvolver sua aplicação para utilizar uma arquitetura orientada para o serviço, você deve fornecer um mecanismo Store-and-Forward . Com esse mecanismo, as mensagens são criadas, armazenadas e finalmente encaminhadas para os respectivos destinos. A implementação mais comum de Store-and-Forward é a fila de mensagem. Essa é a forma na qual produtos orientados para mensagens, como o Microsoft Message Queuing, funcionam. Conforme novas mensagens são criadas, elas são inseridas em filas de mensagem e encaminhadas para os endereços de destino. Ainda que haja alternativas Store-and-Forward (como FTP ou arquivos de cópia entre cliente e servidor), esse guia está focado somente na implementação mais comum: a fila de mensagem.

Seus Smart Clients precisam de uma nova maneira de persistir com as mensagens quando entram para o modo offline. Se sua aplicação necessita criar novas mensagens quando está offline, sua fila deve possuir uma forma de persistir nelas para futuras atualizações com o servidor. A escolha mais óbvia aqui é copiá-las para um disco.

Seu projeto deve incluir uma funcionalidade que garanta que as mensagens sejam entregues com sucesso nos respectivos destinos. Para tanto, o projeto deve levar em consideração os seguintes cenários:

- **Falta de confirmação de que uma mensagem foi adequadamente enviada.** Em geral, você não deve supor que uma mensagem foi recebida no servidor somente por tê-la deixado em uma fila.
- **Perda de conectividade entre o cliente e o servidor.** Em alguns casos, você deve retornar uma mensagem a uma fila pelo fato de a conectividade ter sido perdida entre o cliente e o servidor.
- **Falta de confirmação de um serviço.** Nesse caso, você envia uma confirmação independente para informar o cliente de que a informação foi recebida.

Seu mecanismo Store-and-Forward também necessita suportar uma funcionalidade adicional, como criptografia de mensagem, priorização, bloqueio e sincronização.

Construir e desenvolver uma arquitetura confiável de mensagens é uma tarefa complexa e requer experiência e conhecimentos consideráveis. Por essa razão, você deve considerar produtos comerciais como o Microsoft Message Queuing. Entretanto, o Microsoft Message Queuing requer um software no cliente, o que pode não ser uma opção para todos os Smart Clients.

Outra opção para o gerenciamento de fila de mensagem é usar o Smart Client Offline Application Block, disponível em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/offline-CH01.asp>.

Esse bloco de aplicação oferece serviços e infra-estrutura que os Smart Client utilizam para oferecer capacidades offline para suas aplicações. O bloco suporta a abordagem Store-and-Forward de mensagens utilizando o conceito de fila de mensagem. Por padrão, o bloco suporta integração com o Message Queuing, dentre outros mecanismos de persistência de mensagens, memória, armazenamento isolado, e Microsoft SQL Server™ Desktop Engine (MSDE).

## **Gerenciamento de Conflitos de Regras de Negócios e de Dados**

As alterações realizadas em uma aplicação em modo offline devem ser sincronizadas ou reconciliadas com o servidor ao mesmo tempo. Isso apresenta a possibilidade de um conflito ou outro problema que deve ser solucionado pela aplicação, pelo usuário ou pelo administrador. Quando há conflitos, você deve garantir que eles sejam detectados e resolvidos.

De modo diferente dos conflitos de dados, os conflitos de business rule (regras de negócio) não ocorrem por que há um conflito entre dois dados, mas por que uma business rule foi violada em algum momento e precisa ser corrigida. Ambos os conflitos podem requerer a manipulação pela aplicação do cliente ou pelo usuário.

Como um exemplo de um conflito de business rule, suponha que você tenha uma aplicação de gerenciamento de pedidos que armazena em cache um catálogo de produto, de modo que um usuário possa entrar com pedidos no sistema quando estiver offline. Os pedidos serão, em seguida, encaminhados para o servidor quando a aplicação voltar ao modo on-line. Se um pedido contiver um produto que estava no catálogo de produto armazenado, mas que foi descontinuado à hora que a aplicação voltar ao modo on-line, quando os dados do pedido forem encaminhados para o servidor, ele verificará os detalhes do pedido e verá que o produto foi descontinuado. Nesse momento, a aplicação pode informar o usuário de que há um problema com o pedido. Se o produto em questão foi substituído, o sistema pode dar ao usuário a opção de mudar para um produto diferente. Esse tipo de situação não envolve conflito, pois os dados não entram em conflito com nada, ainda que estejam incorretos e necessitem ser corrigidos.

Ainda que exceções de business rule e de conflitos de dados sejam dois tipos diferentes de exceção, elas podem ser manipuladas, na maioria dos casos, utilizando as mesmas abordagem e infra-estrutura básica. Essa seção discute como manipular conflitos de dados e de business rule em uma aplicação Smart Client.

## **Particionar e Proteger Dados**

Qualquer sistema que permita a múltiplos usuários o acesso a dados compartilhados tem o potencial para produzir conflitos. Ao desenvolver sua aplicação Smart Client, você deve determinar se ela particionará dados e como ela executará a proteção, uma vez que esses fatores ajudam a determinar a probabilidade com que os conflitos irão aparecer em sua aplicação.

### Particionamento de Dados

O particionamento de dados é utilizado em várias situações nas quais indivíduos diferentes tenham controle sobre seções distintas de dados. Por exemplo, um representante de vendas tem um certo número de contas atribuídas somente a ele. Nesse caso, você pode particionar os dados de forma que somente aquele representante de vendas possa alterar aquelas contas. Particionar os dados dessa forma permite aos usuários fazer alterações arbitrárias aos dados sem o medo de encontrar conflitos.

Desenvolver suas aplicações de modo a usar o particionamento de dados é sempre uma opção muito restritiva e, portanto, nem sempre uma boa solução. Contudo, se o particionamento de dados for uma solução prática para uma aplicação específica, você deve considerá-la, uma vez que ela ajuda a reduzir o número de conflitos produzidos pela aplicação.

### Proteção Pessimista

A proteção pessimista envolve o fato de o sistema usar proteções mutuamente exclusivas para garantir que somente um indivíduo opere no sistema de cada vez. Todas as solicitações aos dados são serializadas. Por exemplo, antes de sair à rua, um vendedor pode acessar um banco de dados e verificar as contas dos consumidores de uma determinada área geográfica.

Essa verificação requer a atualização de uma planilha e o envio de mensagens para que outros atualizem o status daquela conta. Agora, quando o vendedor estiver na rua, o resto dos funcionários saberá que ele tem acesso exclusivo aos arquivos daquele consumidor e poderá fazer modificações à vontade. Quando ele voltar ao escritório e sincronizar os novos dados com os dados do servidor, não haverá conflitos. Após sincronizar os dados, o vendedor destravar a proteção lógica.

O principal problema da proteção pessimista é o de que se várias pessoas precisarem operar nos mesmos dados na mesma hora, eles terão que esperar para que os dados estejam disponíveis. Para Smart Clients ocasionalmente conectados, os dados podem ficar protegidos até que um cliente volte ao modo on-line novamente, o que pode levar um longo tempo. Essas razões podem fazer com que a proteção pessimista seja boa em termos de integridade de dados, mas ruim em termos de simultaneidade.

Na realidade, a proteção pessimista é apropriada somente para alguns poucos tipos de aplicações ocasionalmente conectadas. Em sistemas de gerenciamento de documentos, por exemplo, os usuários podem retirar intencionalmente alguns documentos por um período longo de tempo enquanto trabalham com eles. Entretanto, como a escalabilidade e a complexidade aumentam, a proteção pessimista torna-se uma escolha menos prática.

### Proteção Otimista

A maioria das aplicações Smart Client ocasionalmente conectadas utilizam a proteção otimista, que permite que múltiplos indivíduos acessem e operem os mesmos dados ao mesmo tempo, presumindo-se que as alterações feitas aos dados não entrarão em conflito. A proteção otimista permite alta simultaneidade no acesso aos dados, às custas da redução da integridade dos dados. Caso haja conflitos, você precisará de uma estratégia para lidar com eles.

Na maioria dos cenários offline, você precisará utilizar a proteção otimista. Assim, você deve esperar que os conflitos ocorram e estar pronto para reconciliá-los quando isso acontecer.

### Monitoramento Não-confirmado ou Dados Provisórios (Tentative)

Quando seus usuários trabalharem offline, quaisquer alterações nos dados não serão confirmadas como uma alteração no servidor. Somente quando os dados tiverem sido mesclados com o servidor e não houver conflitos eles poderão ser considerados confirmados. É importante monitorar dados não-confirmados. Quando houver a confirmação, os dados podem ser marcados como confirmados e poderão ser utilizados apropriadamente.

Você pode desejar exibir dados não-confirmados na interface de usuário da aplicação com uma cor ou fonte diferente para que o usuário possa ter ciência do caráter provisório deles. Geralmente, suas aplicações não devem permitir que os dados sejam utilizados em mais de uma tarefa até que eles sejam confirmados. Isso evita que dados não-confirmados sejam utilizados em atividades que necessitem de dados confirmados. O uso de dados confirmados não é garantia de não haver conflitos, mas ao menos, a aplicação entenderá que em algum momento o dado foi confirmado e que foi alterado posteriormente por alguém.

### **Manipulação de Dados Desatualizados (Stale)**

Mesmo que os dados não tenham sido alterados, eles podem deixar de ser corretos por não serem mais atuais. Esse tipo de dado é conhecido como desatualizado. Ao desenvolver suas aplicações Smart Client, determine como lidar com dados desatualizados e como evitar que seus Smart Clients usem esses dados. Isso é particularmente importante para Smart Clients ocasionalmente conectados porque os dados estão atualizados quando um cliente ficar on-line pela primeira vez, mas tê-lo tornado desatualizados quando ficar on-line posteriormente. Além disso, um vendedor pode criar um pedido para vários itens em uma sexta-feira utilizando dados válidos mas, se não enviar o pedido ao servidor até a segunda-feira seguinte, o custo desses itens pode ter sido alterado.

---

**Nota:** Se uma solicitação de serviço é enfileirada e estiver pronta para ser enviada quando sua aplicação voltar ao modo on-line, as chances dessa solicitação encontrar um conflito de dados ou uma exceção aumentam de acordo com o tempo que a solicitação ficar na fila. Por exemplo, se você enfileirar uma solicitação de serviço que contenha um pedido para um número de itens e não desejar enviar a solicitação por um longo tempo, os itens de seu pedido podem ser descontinuados ou vendidos.

---

Há uma série de técnicas a serem utilizadas para manipular dados desatualizados. O uso de metadado para descrever a validade dos dados e mostrar quando a validade dos dados irá terminar. Isso evita que dados desatualizados sejam passados para o cliente.

No servidor, opte por verificar quaisquer dados do cliente para determinar se são desatualizados antes de permitir que eles se mesquem aos dados no servidor. Se os dados estiverem desatualizados, você terá a oportunidade de garantir que o cliente faça a atualização dos dados de referência antes de re-enviar os dados ao servidor.

O risco de dados desatualizados é maior com aplicações ocasionalmente conectadas do que com aplicações que estejam sempre conectadas. Por essa razão, suas aplicações Smart Client sempre irão executar etapas adicionais de validação para garantir que os dados sejam válidos. Adicionando uma validação extra ao sistema, você também garante que seus serviços sejam mais tolerantes a dados desatualizados e, em alguns casos, você também pode ser capaz de manipular automaticamente a reconciliação no servidor (ou seja, mapear a transação para uma nova conta). Às vezes, mensagens desatualizadas são inevitáveis. A forma com que você lida com esses dados deve pautar-se pelas regras do negócio que está modelando. Em algumas instâncias, dados desatualizados são aceitáveis. Por exemplo, suponha que um pedido seja feito para um determinado item em um catálogo on-line. O item tem um número de catálogo que se tornou desatualizado porque o catálogo on-line mudou.

Entretanto, o item ainda está disponível e não mudou, a alteração no número do catálogo não tem efeito no sistema e o pedido correto é gerado.

Por outro lado, se você estiver fazendo uma transação monetária entre duas contas e uma das contas tiver sido encerrada, você não poderá efetuar a transação. Aqui, a desatualização dos dados importa.

Uma boa regra geral é ter objetos de negócios manipulando dados desatualizados para você. Seus objetos de negócios podem validar os dados que sejam atuais e, se não o forem, podem não fazer nada, podem reconciliar os dados desatualizados com os dados atuais equivalentes, passar a informação de volta ao cliente para ser atualizada ou usar regras de negócios para automatizar uma resposta apropriada.

A reconciliação de dados pode ocorrer no cliente, no servidor ou em ambos. Manipular a reconciliação no servidor permite à sua aplicação detectar prontamente um conflito. Manipular a reconciliação no cliente sobrecarrega algumas das responsabilidades do usuário ou administrador, que podem ser solicitados a resolver manualmente os conflitos.

Não há um melhor método para manipular dados desatualizados. Suas regras de negócios podem ditar que o servidor seja o melhor lugar para manipular dados desatualizados se o cliente não puder solucionar o conflito. Se o servidor não tiver informações suficientes para manipular automaticamente a situação, pode ser necessário solicitar que o cliente limpe seus dados antes de sincronizá-los com o servidor. Da mesma forma, você pode decidir que os dados desatualizados sejam bons para sua aplicação e nesse caso não há nada com que você deva se preocupar.

### **Reconciliar Conflitos**

Ao examinar os requisitos de reconciliação de dados para sua organização, você deve considerar a forma com que a organização trabalha. Em alguns casos, os conflitos são improváveis porque indivíduos diferentes são responsáveis por elementos diferentes de dados. Em outros casos, os conflitos irão existir com maior frequência e você deve garantir que tenha mecanismos para lidar com eles.

Não importando quais precauções você tome, é provável que um cliente envie dados para um serviço de rede que resultem em uma violação de regras de negócios ou conflito de dados. Quando um conflito ocorre, o serviço remoto deve oferecer o maior número possível de detalhes sobre o conflito. Em alguns casos que o conflito de dados não seja um grande problema e possa ser manipulado automaticamente pela aplicação ou pelo servidor. Por exemplo, imagine um sistema de gerenciamento de relacionamento com o cliente (CRM) no qual o usuário altere um número de telefone de consumidor. Quando a alteração é atualizada no servidor, descobre-se que outro usuário também modificou o número de telefone. Você opta por desenvolver seu sistema de forma com que a última alteração seja sempre a prevalecente ou, então, enviar o conflito para um administrador. Se o administrador souber quem fez a alteração e quando, ele pode tomar a decisão sobre qual manter. O importante é que o servidor e a aplicação forneçam suficientes informações detalhadas para habilitar a manipulação automática ou oferecer informações suficientes para um usuário ou administrador para que ele possa reconciliar o conflito.

A reconciliação de dados pode ser um problema complicado e dependente do cenário. Todo negócio e toda aplicação terá regras, requisitos e premissas ligeiramente diferentes. Entretanto, você tem três opções gerais para reconciliação de dados:

- **Reconciliação automática de dados no servidor**
- **Reconciliação personalizada no cliente**
- **Reconciliação terceirizada**

É útil examinar cada um dos três itens.

### **Reconciliação Automática de Dados no Servidor**

Em alguns casos, desenvolva sua aplicação de forma que o servidor use regras de negócios e processos automatizados para manipular conflitos sem afetar o cliente. Você certifica que a última alteração sempre prevaleça, que haja a fusão entre os dois elementos de dados ou utilize uma lógica business mais complexa.

Manipular conflitos no servidor é bom para o aproveitamento e evita que o usuário fique profundamente envolvido, ou seja, perturbado pelo processo de reconciliação. Você deve manter o cliente sempre informado sobre qualquer ação de reconciliação tomada; por exemplo, retornando um relatório de reconciliação para o cliente, explicando o conflito e como ele foi resolvido. Isso permite que o cliente mantenha os dados locais consistentes e informa o usuário sobre o futuro da reconciliação.

Por exemplo, suponha que uma aplicação permita aos usuários entrar com informações de pedido para itens em um catálogo que seja armazenado localmente em cache. Se o usuário pedir um item que tenha sido descontinuado mas, substituído por um modelo similar mais novo, o serviço de pedido pode optar por substituir o item original pelo mais recente. O cliente será, então, informado sobre a alteração para que ele possa modificar o estado local de maneira apropriada.

### Reconciliação Personalizada no Cliente

Em alguns casos, o cliente é o melhor lugar para executar a reconciliação porque ele sabe mais a respeito do contexto da solicitação original. A aplicação pode ser capaz de resolver o conflito automaticamente. Em outros casos, o usuário ou um administrador pode determinar como um conflito deve ser resolvido.

Para garantir a reconciliação efetiva por parte do cliente, o serviço deve enviar ao cliente dados suficientes para que o cliente tome uma decisão inteligente no tocante a como solucionar o conflito. Os detalhes exatos do conflito devem ser reportados de volta para o cliente de modo que ele ou o usuário ou um administrador possam determinar a melhor forma de resolver o problema.

### Reconciliação Terceirizada

Em alguns casos, a escolha será que um terceiro reconcilie conflitos de dados. Por exemplo, um administrador ou supervisor pode ser solicitado a reconciliar conflitos importantes de dados. Eles são os únicos usuários com autoridade para determinar o curso certo de ação. Nesse caso, o cliente precisa ser informado de que a decisão está pendente. O cliente tem a capacidade de continuar usando valores provisórios, mas de maneira geral ele terá que esperar até que o conflito subjacente tenha sido resolvido. Quando isso acontecer, o cliente será informado. De outro modo, o cliente pode examinar periodicamente para determinar o status e continuar quando receber o valor reconciliado.

## Interação com Web Services CRUD

Muitos Web Services são criados com interfaces CRUD (Create, Read, Update, Delete). Essa seção cobre várias estratégias para criar aplicações ocasionalmente conectadas que consumam tais serviços.

### Create (Criar)

Criar registros deveria ser uma tarefa relativamente simples em um serviço da Web CRUD, contanto que você gerenciasse corretamente a criação dos registros. O ponto mais importante é identificar de maneira única cada registro criado. Na maioria das situações, você faz isso utilizando um identificador único como a chave primária em seus registros. A seguir, mesmo que dois registros aparentemente idênticos sejam criados em clientes separados, os registros serão vistos como diferentes quando houver a merge replication.

---

**Nota:** Em alguns casos, você pode desejar que os registros não sejam tratados como únicos. Nesses casos, você pode gerar uma exceção quando os dois registros entrarem em conflito.

---

Há vários métodos para criar identificadores únicos em um cliente offline, incluindo:

- Envio do registro como um objeto de transferência de dados (DTO) sem ID único e permitindo que o servidor atribua o ID.
- Uso de um identificador globalmente único (GUID) que pode ser atribuído pelo cliente, como um **System.Guid**.
- Atribuição de um ID temporário no cliente e a substituição pelo ID real no servidor.
- Atribuição de um bloco de IDs únicos para cada cliente.
- Uso do nome do usuário ou ID para prefixar todos os IDs alocados e incrementá-los no cliente de modo que elas sejam globalmente únicas por padrão.



### Read (Ler)

Não há conflitos de dados com operações de leitura, uma vez que elas são, por definição, operações somente-leitura. Entretanto, os problemas ainda podem ocorrer com operações de leitura em Smart Client ocasionalmente conectados. Você deve armazenar em cache quaisquer dados que precisem ser lidos no cliente antes de ele entrar em modo offline. Esses dados podem tornar-se desatualizados antes de o cliente voltar ao modo on-line, levando à ocorrência de dados incorretos no cliente e problemas quando houver a sincronização no servidor. Para maiores informações sobre como lidar com dados desatualizados, consulte "Manipulação de Dados Desatualizados" mais acima, nesse capítulo.

### Update (Atualizar)

As atualizações de dados levam, freqüentemente, a conflitos de dados porque múltiplos usuários podem atualizar os mesmos dados, levando a conflitos quando ocorrer a merge replication. Você utiliza uma série de métodos para minimizar a ocorrência de conflitos e resolvê-los quando ocorrerem. Para maiores informações, consulte "Gerenciamento de Conflitos de Business Rules e de Dados" mais acima, nesse capítulo.

### Delete (Excluir)

Excluir um registro é uma operação pontual pois um registro só pode ser excluído uma vez. Tentar excluir o mesmo registro duas vezes não produzirá nenhum efeito no sistema. Entretanto, há algumas coisas que você deve ter em mente ao desenvolver sua aplicação e serviço da Web para manejar exclusões. Primeiramente, você deve marcar os registros como provisoriamente excluídos no cliente e, a seguir, enfileirar as solicitações de exclusão no servidor. Isso significa que, se o servidor não for capaz de excluir um registro por alguma razão, a exclusão poderá ser desfeita no cliente.

Assim como acontece ao criar registros, você também precisa certificar-se de que esteja se referindo aos registros utilizando um identificador único. Isso assegurará que você sempre excluirá o registro correto no servidor.

## Uso de uma Abordagem Baseada em Tarefa

A abordagem baseada em tarefa utiliza um objeto para encapsular uma unidade de trabalho como sendo uma tarefa de usuário. O objeto **Tarefa** é responsável por tomar conta das interações de interface de usuário, estado e serviço necessários para que o usuário complete uma tarefa específica. A abordagem baseada em tarefa é particularmente útil quando você desenvolve e constrói aplicações Smart Client com capacidade offline, uma vez que ela permite que você encapsule os detalhes do comportamento offline em um único lugar. Isso permite à interface de usuário focar-se em questões relacionadas à UI, mais do que em lógica de processamento. A granularidade e detalhes de suas tarefas depende do cenário exato da aplicação. Alguns exemplos de tarefas são:

- Registro de informações de pedidos
- Alterações de detalhes de contatos de consumidores
- Composição e envio de mensagens eletrônicas
- Atualização de status de pedido

Para cada uma dessas tarefas, um objeto **Tarefa** é criado e usado para guiar o usuário pelo processo, armazenar todo estado necessário, interagir com a interface do usuário e com quaisquer outros serviços necessários.

Quando uma aplicação estiver operando offline, ela precisará enfileirar solicitações de serviço e fazer possíveis alterações locais de estado utilizando valores provisórios ou não-confirmados. Durante a sincronização, a aplicação precisa executar a solicitação atual de serviço e fazer possíveis alterações



adicionais locais de estado para confirmar o sucesso da solicitação de serviço. Ao encapsular os detalhes desse processo dentro de um único objeto **Tarefa**, o que insere a solicitação na fila e monitora alterações de estado provisórias e confirmadas, simplifique o desenvolvimento da aplicação, proteger contra alterações de implementação e permitir que todas as tarefas sejam manipuladas de uma forma padrão. O objeto Tarefa pode oferecer informações detalhadas sobre o estado da tarefa por meio de várias propriedades e eventos, incluindo:

- **Status pendente.** Indica que a tarefa está pendente de sincronização.
- **Status confirmado.** Indica que a tarefa foi sincronizada e confirmada como bem-sucedida.
- **Status de conflito.** Indica que um erro ocorreu durante a sincronização. Outras propriedades irão fornecer detalhes sobre o conflito ou erro.
- **Concluído.** Indica a porcentagem de conclusão ou marca a tarefa como concluída.
- **Disponibilidade da tarefa.** Algumas tarefas não estarão disponíveis quando a aplicação estiver on-line ou offline, ou, se a tarefa for parte de um workflow ou processo de interface de usuário, ela pode não estar disponível até que uma tarefa pré-requisito tenha sido concluída. Essa propriedade pode ser ligada a flags habilitadas para itens de menu ou botões da barra de ferramentas para evitar que o usuário inicie tarefas inapropriadas.

Outro benefício da abordagem baseada em tarefa é que ela foca a aplicação nos usuários e suas tarefas, o que pode resultar em uma aplicação mais intuitiva.

## Manipulação de Dependências

Se uma tarefa de usuário envolve mais de uma solicitação de serviço, a tarefa precisa ser manipulada muito cuidadosamente de modo que o usuário possa concluir toda a tarefa quando estiver offline. O desafio é que as solicitações de serviço são freqüentemente dependentes umas das outras. Por exemplo, suponha que você tenha uma aplicação que permita que consumidores possam fazer reservas para as férias. Para fazer uma reserva, a aplicação utiliza uma série de serviços para executar cada parte da tarefa na seguinte seqüência:

1. Reservar um carro.
2. Reservar hotel.
3. Comprar passagens aéreas.
4. Enviar confirmação por e-mail.

Cada um desses serviços precisa ser implementado por sistemas diferentes, talvez até por companhias diferentes. Em um mundo perfeito, cada solicitação de serviço seria feita sem problemas todas as vezes, de modo tal que seu usuário pudesse reservar o carro, reservar o hotel, e comprar as passagens e, ainda, receber uma confirmação por e-mail de que tudo saiu a contento. Entretanto, nem todas as solicitações de serviço são perfeitas, e sua aplicação deve ser capaz de resolver condições de erro e manipular business rules que afetem o funcionamento de toda a tarefa. Escrever o código para esse tipo de tarefa é extremamente desafiador, uma vez que cada parte da tarefa (ou seja, cada solicitação de serviço para um serviço específico) depende de outra parte da tarefa.

As dependências podem, por si só, depender de lógica business complexa, o que, mais adiante, complicará a lógica por trás de toda a tarefa. Por exemplo, sua aplicação de reserva de férias pode permitir que as férias sejam reservadas mesmo que não haja carros disponíveis, contanto que as reservas de hotel e vôo possam ser feitas. As dependências entre solicitações individuais de serviço podem ser tanto *diretas* quanto *reversas*:

- **Dependências diretas.** Se, durante a sincronização, a primeira solicitação obtiver êxito, mas uma solicitação subsequente falhar, você poderá ter que reverter a primeira solicitação através de uma transação de compensação. Esse requisito pode adicionar uma complexidade significativa à aplicação.
- **Dependências reversas.** Se uma aplicação estiver operando offline e enviar uma solicitação de serviço como parte de uma tarefa de solicitação multi-serviços, ela terá que assumir que a solicitação será concluída com êxito de forma que ela possa enfileirar solicitações subsequentes e não evitar que o usuário conclua a tarefa. Nesse caso, todas as solicitações posteriores são dependentes do sucesso da primeira. Se a primeira solicitação falhar durante a sincronização, a aplicação deve ter ciência de que todas as solicitações posteriores deverão ser excluídas ou ignoradas.

## Manipulação de Dependências no Servidor

Para reduzir as complexidades associadas com as dependências entre solicitações de serviço, o serviço da Web deve oferecer uma única solicitação de serviço por tarefa de usuário. Isso permite que o usuário conclua uma tarefa que será manipulada durante a fase de sincronização como uma única solicitação atômica para o serviço da Web. Uma única solicitação atômica elimina a necessidade de monitorar dependências de solicitação de serviço, o que pode complicar significativamente a implementação da aplicação do lado do cliente ou do servidor.

Por exemplo, em vez de escrever suas três interfaces de serviço como três etapas separadas:

```
BookCar()  
  
BookHotel()  
  
BookAirlineTickets()
```

Você pode combiná-las em uma única etapa:

```
BookVacation( Car car, Hotel hotel, Tickets airlineTickets )
```

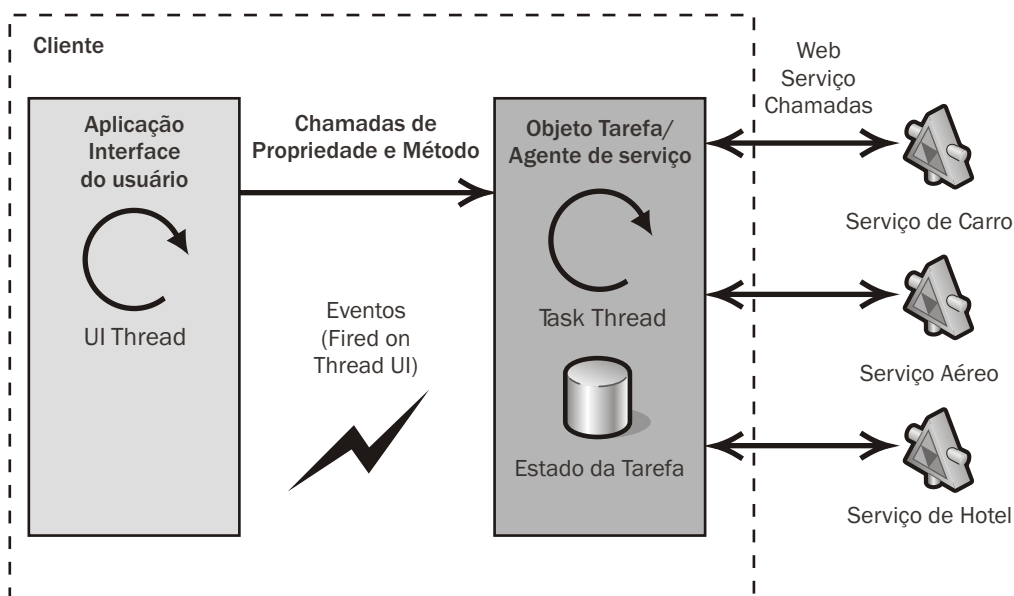
Combinar as etapas dessa forma significa que, agora, você tem uma interação atômica em vez de três separadas. No exemplo, o serviço da Web BookVacation (Reserva de férias) seria responsável pela coordenação necessária entre os elementos que perfazem o serviço.

## Manipulação de Dependências no Cliente

Você também pode monitorar as dependências de solicitação de serviço no cliente. Essa abordagem oferece uma flexibilidade significativa, e permite que o cliente controle a coordenação entre quaisquer números de serviços. Entretanto, essa abordagem é de difícil desenvolvimento e teste. A abordagem baseada em tarefa é uma boa forma de monitorar dependências de solicitação de serviço no cliente, e fornece uma forma de encapsular tudo que for necessário da lógica business e manipulação de erro em um único lugar, o que simplifica o desenvolvimento e o teste. Para maiores informações sobre a abordagem baseada em tarefa, consulte "Uso de uma Abordagem Baseada em Tarefa", mais acima, nesse capítulo.

Por exemplo, o objeto **Tarefa** usado para reservar férias saberia que ele teria que executar três solicitações de serviço. Ele implementaria a lógica business necessária de modo que pudesse controlar as solicitações de serviço apropriadamente se uma condição de erro fosse encontrada. Se a chamada de serviço **BookCar** (Reservar carro) falhasse, ele seguiria com as chamadas de serviço para **BookHotel** (Reservar hotel) e **BookAirlineTickets** (Reservar passagens aéreas). Se essa última

falhasse, ele seria, então, responsável pelo cancelamento de qualquer reserva de carro ou hotel, criando uma solicitação de serviço de transação de compensação para cada serviço. A Figura 4.2 ilustra essa abordagem baseada em tarefa.



**Figura 4.2**

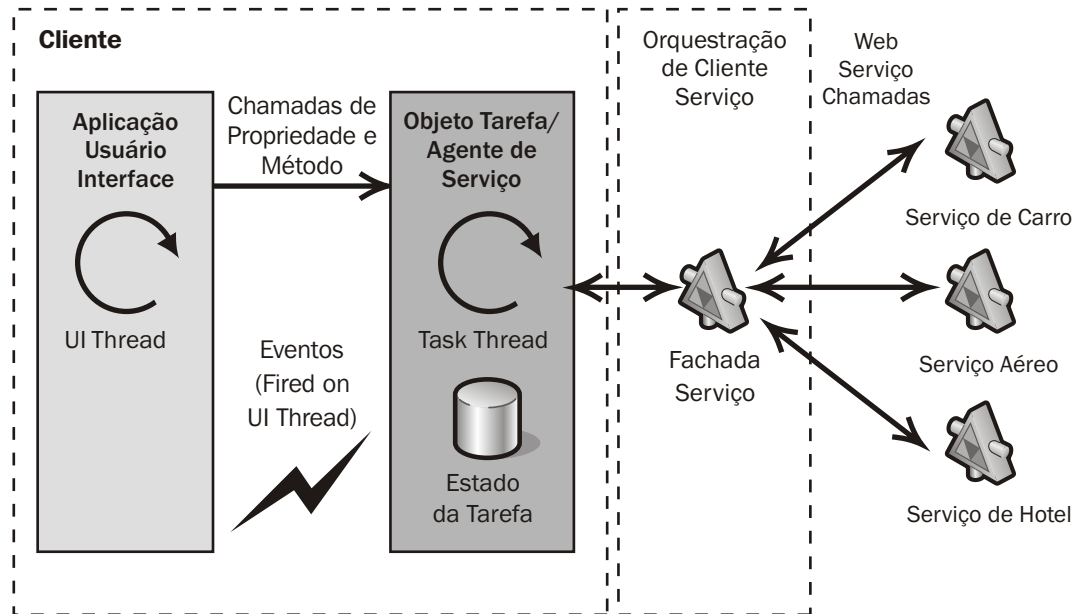
*Abordagem baseada em tarefa com inter-dependências*

## Uso da Orquestração Middleware

Às vezes, as dependências e regras de negócios correspondentes em suas aplicações são suficientemente complexas para requerer alguma forma de orquestração middleware, como

Microsoft BizTalk® Server, que coordena as interações entre múltiplos Web Services e uma aplicação de cliente. A orquestração middleware está localizada na camada intermediária e oferece um serviço da Web de fachada para interagir com o Smart Client.

O serviço da Web de fachada apresenta uma interface apropriada, específica para a aplicação, que permite uma única solicitação Web por tarefa de usuário. Quando uma solicitação de serviço é recebida, o serviço de orquestração processa a solicitação iniciando e coordenando as chamadas para os Web Services necessários, possivelmente agregando os resultados antes de retorná-los para o cliente. Essa abordagem oferece uma maneira mais escalável de dar conta das interações entre múltiplos serviços da Web. O BizTalk também oferece serviços importantes, como transformação de dados e mecanismo de business rules que podem ajudar significativamente ao interagir com Web Services distintos ou sistemas de legado e em cenários business complexos. Além disso, essa abordagem fornece garantias importantes de disponibilidade e confiança, o que ajuda a garantir a consistência entre os múltiplos serviços. A Figura 4.3 ilustra o uso da orquestração middleware.

**Figura 4.3**

*Orquestração middleware usada para coordenar dependências de serviço*

## Sumário

Alguns clientes precisam operar com eficiência quando conectados e desconectados da rede. Ao desenvolver seus Smart Clients, você precisa certificar-se de que eles podem funcionar efetivamente em ambas situações, e transitar sem problemas entre as duas.

Há duas estratégias amplas para desenvolver comunicações Smart Client: orientada para o serviço e centrada nos dados. Após determinar qual dessas duas usar, você precisará tomar algumas decisões fundamentais de design para permitir que seus Smart Clients trabalhem em modo offline. Na maioria dos casos, os clientes devem ser desenvolvidos para utilizar a comunicação assíncrona e interações simples de rede. Os clientes precisarão armazenar dados em cache para uso quando estiverem offline, e você precisará de um método para manipular conflitos de dados e de business rules quando o cliente voltar ao modo on-line. Em muitos casos, clientes offline permitem que os usuários executem uma série de tarefas que são dependentes umas das outras. Será necessário lidar com essas dependências no evento em que uma dessas tarefas falhe quando alcançar o servidor. Seus Smart Clients podem, também, precisar interagir com Web Services de padrão CRUD.

A abordagem baseada em tarefa pode simplificar dramaticamente o processo de aplicações offline. Considere a opção de implementar essa abordagem em seus Smart Clients, ela também pode oferecer uma forma efetiva de manipulação de dependências, tanto no servidor quanto no cliente.

# 5

## Considerações de Segurança

Os Smart Clients são aplicações distribuídas que, geralmente, espalham produtos e tecnologias diferentes. Gerenciar a segurança nessas aplicações pode ser uma tarefa muito desafiadora. No servidor, você precisa adotar uma abordagem de proteção da rede, do próprio servidor e, a seguir, da aplicação. No cliente, você deve concentrar-se no uso de recursos de segurança da plataforma (incluindo o sistema operacional e o Microsoft®.NET Framework), nas operações privilegiadas que o código de cliente pode executar (segurança de acesso ao código) e nas interações com a plataforma do servidor (domínio) e aplicação do servidor.

A segurança efetiva depende de uma abordagem de defesa em profundidade. Ao proteger seus Smart Clients, é muito importante considerar todos os aspectos de segurança, incluindo:

- **Autenticação.** Essa operação identifica de maneira única o usuário da aplicação cliente de forma que somente usuários aprovados possam acessar toda a aplicação ou parte dela.
- **Autorização.** Isso determina as operações que o usuário unicamente identificado pode executar. Essas operações podem ser tarefas ou operações em recursos aos quais o usuário autenticado tenha acesso.
- **Validação de dados.** Isso é uma garantia de que somente dados válidos e apropriados são aceitos pela aplicação. Se você permitir qualquer registro sem validar os dados primeiro, poderá estar disponível a ataques por meio da entrada de dados maliciosos.
- **Proteção de dados sensíveis.** Isso significa que o armazenamento de dados sensíveis (como senhas e dados confidenciais) e a transmissão pela aplicação são seguros. A criptografia de dados sensíveis garante que os dados não estarão disponíveis em texto simples; dependendo da escolha do algoritmo, ele também pode garantir que a informação não será manuseada para manter a integridade.
- **Auditoria e log-in.** Isso envolve manter um registro dos eventos e das ações dos usuários. Você deve considerar fazer o log-in de ações ou atividades-chave de usuários no servidor ou logá-las de maneira segura no cliente, uma vez que registros de log em computadores de cliente podem ser mexidos ou limpos.
- **Gerenciamento de exceção.** Isso garante que a aplicação lide com exceções de maneira apropriada. Os detalhes das exceções podem ser registrados no log de registro ou no log da aplicação.

- **Gerenciamento de alteração de configuração.** Garante que você monitore a configuração de seu ambiente de TI e quaisquer alterações que ocorram nele. Ao fazer isso, você poderá ver se houve qualquer alteração não-autorizada e determinar as implicações de segurança de quaisquer alterações autorizadas.

Esse capítulo examina em detalhes alguns dos principais desafios-chave de segurança que você encontrará ao desenvolver suas aplicações Smart Client, especialmente focando em autenticação, autorização, validação de dados, e proteção de dados sensíveis. O capítulo cobre a segurança do acesso ao código, a tecnologia-chave no .NET Framework para gerenciar a segurança em nível de código e não em nível de usuário.

Outro aspecto importante que você deve levar em consideração ao examinar a segurança do Smart Client é a forma com que seu Smart Client é utilizado. Para maiores informações sobre questões de segurança que afetem o uso, consulte o capítulo 7, "Uso e Atualização de Aplicações Smart Client".

Nota: Qualquer código utilizado em sua aplicação deve ser analisado com o FxCop. Essa ferramenta permite a você verificar módulos gerenciados de códigos para conformidade com as diretrizes de design do .NET Framework, incluindo um nível base de conformidade de segurança. O FxCop pode ser baixado do site GotDotNet em <http://www.gotdotnet.com/team/fxcop/>.

## Autenticação

A autenticação é o processo compreendido pela identificação única de um usuário por meio da verificação de suas credenciais. A autenticação de usuário pode ser requerida quando o usuário tentar executar ou instalar a aplicação ou quando a aplicação estabelecer uma conexão a um serviço remoto ou acessar dados mantido localmente.

Essa seção examina alguns cenários de autenticação comuns a Smart Clients, cobre alguns dos diferentes modos pelos quais você pode fazer chamadas de rede autenticadas e discute como recolher credenciais de usuários e validá-las quando em modo offline.

### Cenários de Autenticação Smart Client

Dependendo do estilo e funcionalidade da sua aplicação Smart Client, você pode precisar autenticar o usuário em um ou mais pontos durante a interação do usuário com a aplicação. Há quatro pontos nos quais você desejáveis para autenticar o usuário:

- Quando a aplicação é instalada.
- Quando a aplicação é executada.
- Quando o usuário acessa dados mantidos localmente.
- Quando o usuário acessa serviços externos da rede.

### Instalação

Se sua aplicação é implementada centralmente (por exemplo, usando implementação no-touch), você protege a aplicação no servidor Web de forma que ela será instalada apenas por usuários autorizados. Esses usuários devem, primeiramente, acessar a aplicação e fazer o download para os computadores cliente.

Proteger o acesso a uma aplicação Smart Client implementada de modo no-touch é um procedimento similar ao de proteção de qualquer outro artefato localizado no servidor Web, como uma página Web. Os Serviços de Informações da Internet da Microsoft (Microsoft Internet Information Services-IIS) fornecem uma série de mecanismos de autenticação, como integrada do Windows, digest, ou autenticação básica.

---

**Nota:** As autenticações básica e digest não são apropriadas se você estiver usando uma implementação no-touch e se sua aplicação utilizar um arquivo de configuração para armazenar as definições de configuração, uma vez que o arquivo de configuração não pode ser baixado automaticamente pelo .NET Framework usando esses mecanismos.

---

Após selecionar o mecanismo apropriado de autenticação, e o IIS identificar o usuário a partir das credenciais, você protege a aplicação e os módulos dependentes definindo permissões de arquivos na aplicação e nos arquivos do módulo. Para facilitar o gerenciamento de uma grande quantidade de usuários, considerar a possibilidade de oferecer acesso a um Microsoft Windows® group (por exemplo, SmartClientAppUsers) e inserir usuários individuais no grupo.

Todos os usuários que você precisar autenticar devem ter uma identidade Windows no servidor para que o IIS possa proteger o acesso à aplicação e seus módulos, mas eles não precisam necessariamente estar logados aos computadores clientes usando essa identidade. Se a conta de logon do usuário não for conhecida pelo servidor, o usuário é solicitado a fornecer um nome de usuário e senha quando clicar no link para o arquivo executável da aplicação.

Se usar a autenticação integrada do Windows, as credenciais do usuário logado são automaticamente utilizadas para tentar ganhar acesso novamente à aplicação. Isso permitirá acesso seguro e contínuo quando o usuário estiver logado com uma identidade comum tanto ao cliente quanto ao servidor.

### **Acesso Autenticado à Aplicação**

Autenticar usuários para instalar uma aplicação garante que somente os usuários autenticados e autorizados possam ser capazes de executar sua aplicação de um local central. Entretanto, pelo fato de a aplicação e seus artefatos dependentes poderem ter sido armazenados em cache no computador cliente, você não pode confiar em seu mecanismo para autenticar o usuário toda vez que a aplicação for executada. Nesse caso, ou quando a aplicação for implementada localmente, você precisa considerar cuidadosamente como a autenticação de usuário deve ser feita pela sua aplicação. Você necessita autenticar os usuários toda vez que eles executarem a aplicação, principalmente se sua aplicação oferecer funcionalidade sensível ou se você precisar ser capaz de revogar a autorização de um usuário para executar a aplicação a qualquer momento.

Nos casos em que o usuário tenha logado ao computador cliente utilizando uma identidade comum tanto ao cliente quanto ao servidor, você será capaz de confiar no fato de que o usuário pôde logar no computador cliente como autenticação suficiente para executar a aplicação. Essa abordagem permite a você usar o sistema operacional Microsoft Windows para oferecer autenticação de usuário, eliminando a necessidade de implementar isso no código. Além disso, pelo fato de o Windows poder armazenar, em cache, as credenciais de usuários em modo offline, você não precisará armazená-las.

Para computadores cliente nos quais você não tenha qualquer controle sobre acesso de usuários, como aqueles de fora da intranet de sua organização, adote um mecanismo de autenticação personalizada para angariar as credenciais de usuário e autenticá-las contra uma autoridade de segurança remota. Se a aplicação for capaz de operar offline, será necessário armazenar em cache credenciais válidas no cliente, de modo que a re-autenticar o usuário contra elas quando ele iniciar a aplicação. Você deve reforçar a re-autenticação on-line periodicamente para evitar o uso ilimitado de tais aplicações.

### **Acesso Autenticado aos Dados Locais**

Uma aplicação Smart Client geralmente armazena, em cache, dados obtidos de um serviço remoto (por exemplo, para aumentar a resposta ou permitir capacidades offline). Se os dados forem sensíveis, considere a possibilidade de autenticar o usuário antes de fornecer acesso a ele. Nesse caso, você opte por permitir que usuários não-autenticados executem a aplicação, mas requerer que os usuários sejam totalmente autenticados e autorizados antes de fornecer acesso a dados sensíveis.

---

**Nota:** É importante garantir que somente os dados aos quais o usuário esteja autorizado a acessar sejam armazenados localmente. Se os dados forem sensíveis, você também precisará certificar-se de que medidas adequadas sejam tomadas para garantir a segurança. Para detalhes, consulte: "Manipulação de Dados Sensíveis" mais adiante nesse capítulo.

---

Dados mantidos localmente devem ser mantidos em um lugar seguro e criptografados. Independentemente de como os usuários sejam autenticados, você geralmente irá desejar usar as credenciais de alguma forma para acessar e proteger os dados.

Utilize as credenciais padrão que foram utilizadas para fazer login no computador cliente, ou poderá precisar obter credenciais personalizadas para autenticar o usuário contra uma autoridade de segurança remota. A primeira possibilidade é mais apropriada para aplicações que são executadas em uma situação de intranet, ao passo que a segunda opção é apropriada para aplicações executadas em uma situação de Internet ou extranet, onde os usuários não estejam geralmente no mesmo domínio dos serviços remotos por eles acessados. Um dos benefícios do uso da autenticação integrada é o de que o sistema operacional autentica o usuário, protege a aplicação e dados locais e pode armazenar as credenciais de usuário quando o usuário estiver offline.

Para maiores informações sobre como armazenar dados sensíveis localmente, consulte "Manipulação de Dados Sensíveis" mais adiante, nesse capítulo.

### **Acesso Autenticado à Rede**

Você pode optar por permitir o acesso anônimo à aplicação e permitir que qualquer usuário faça o download e execute-a. Entretanto, após a aplicação ser executada no cliente, ela acessa serviços remotos na rede, como um serviço da Web, a fim de obter dados e serviços.

O acesso a dados e serviços na rede necessitam ser protegido para evitar o acesso não autorizado. Há muitas formas de proteger o acesso ao serviço remoto, mas você normalmente precisa passar as credenciais de usuário para o serviço remoto para que ele execute a autenticação de usuário.

Autenticar usuários quando eles tiverem acesso a serviços remotos na rede é uma questão importante. Algumas das opções para garantir chamadas autenticadas de serviço na rede são descritas em maior profundidade em "Tipos de Autenticação de Acesso de Rede" mais adiante nesse capítulo.

### **Escolher o Modelo Certo de Autenticação**

A seção anterior descreveu quatro estágios nos quais você pode optar por autenticar o usuário. Você pode optar por autenticar o usuário em um ou mais desses pontos, dependendo da natureza da sua aplicação e de sua funcionalidade. É importante escolher o ponto certo para ajudar a garantir que sua aplicação e dados permaneçam seguros, enquanto minimiza qualquer impacto no aproveitamento de sua aplicação. Se sua aplicação for implementada centralmente (por exemplo, se for implementada usando o método no-touch ou se for implementada em um arquivo compartilhado), você pode optar por restringir o acesso aos usuários que não estejam autorizados. Caso desejar que sua aplicação esteja disponível para qualquer um que queira usá-la, não será necessário autenticar o usuário quando a aplicação for instalada.

Os computadores cliente, via de regra, não são fisicamente seguros e podem, até mesmo, ser acessíveis publicamente. Se esse for o caso, e sua aplicação fornecer funcionalidade sensível, você precisará autenticar o usuário frequentemente quando a aplicação for executada. Se sua aplicação fornecer funcionalidade genérica que esteja disponível para usuários anônimos, você não precisará autenticar o usuário nesse ponto. Entretanto, opte por fornecer uma porção das funcionalidades de sua aplicação para usuários anônimos, mas requerer autenticação antes de permitir que eles acessem funcionalidades mais restritas. Proteger o acesso a dados sensíveis mantidos localmente é muito importante. Se sua aplicação for implementada em um equipamento que não seja fisicamente protegido ou que seja acessível ao público, os dados sensíveis devem ser protegidos e devem ser



acessíveis apenas para usuários autenticados e autorizados. Sua aplicação poderá fornecer funcionalidades genéricas para usuários anônimos, mas requerer autenticação de usuário quando os usuários tentarem acessar dados sensíveis.

O uso da autenticação integrada do Windows tem benefícios quando a aplicação for executada em modo offline. Nesse caso, o Windows armazena as credenciais do usuário de modo que o usuário seja autenticado quando logar-se ao computador cliente em modo offline. Esse comportamento elimina a necessidade de seu cliente autenticar o usuário caso você precise da autenticação de usuário quando a aplicação for executada ou acessar dados sensíveis mantidos localmente.

## Tipos de Autenticação de Acesso de Rede

Há muitas tecnologias diferentes que você pode usar para autenticar os usuários ao acessar serviços remotos, incluindo:

- **Autenticação integrada do Windows**
- **Autenticação básica HTTP**
- **Autenticação digest HTTP**
- **Autenticação baseada em certificado**
- **Autenticação baseada em Web Services Enhancements (WSE)**
- **Autenticação personalizada**

### Autenticação integrada do Windows

Com a autenticação integrada do Windows, o usuário é autenticado ao logar no computador utilizando uma conta do Windows válida. As credenciais podem ser uma conta local (uma conta local no computador) ou uma conta de domínio (um membro válido de um domínio do Windows). A identidade estabelecida durante o login é usada de forma transparente pela aplicação para acessar recursos para toda a duração da sessão do usuário. Isso significa que suas aplicações podem fornecer acesso contínuo a recursos de rede, como serviços da Web, de um modo seguro, sem ter que solicitar ao usuário credenciais adicionais ou mesmo repetidas.

---

**Nota:** O acesso aos recursos da rede será contínuo somente se os recursos de rede também utilizarem a autenticação integrada do Windows.

---

A autenticação integrada do Windows utiliza um de dois protocolos de autenticação: Kerberos ou NTLM. Essas tecnologias não passam um nome de usuário e uma senha pela rede para autenticar ou validar o usuário. Como resultado, sua aplicação ou infra-estrutura não tem que fornecer segurança adicional para gerenciar credenciais durante o trânsito.

Aplicações cliente que confiam na segurança do Windows utilizam uma implementação da interface **Identity** chamada **WindowsIdentity**.

---

**Nota:** O .NET Framework também fornece a interface intimamente relacionada **IPrincipal**. Para maiores informações sobre as interfaces **Identity** e **IPrincipal**, consulte "Autorização", mais adiante nesse capítulo.

---

### Web Services que Utilizam a Autenticação Integrada do Windows

Para Web Services que sejam configurados para a autenticação integrada do Windows, a aplicação cliente pode fornecer credenciais de usuário atualmente logado para fins de autenticação antes de fazer chamadas a serviço da Web. Ao adicionar uma referência a um serviço da Web em sua aplicação a partir de dentro do sistema de desenvolvimento Microsoft Visual Studio® .NET, uma classe proxy é gerada automaticamente e adicionada a seu projeto de modo a acessar, de maneira programática, o serviço da Web. O código a seguir ilustra como configurar as credenciais do usuário que está atualmente logado.

```
MyService service = new MyService(); // A proxy for a web service.
service.Credentials = CredentialCache.DefaultCredentials;
service.SomeServiceMethod();          // Call the web service.
```

Nesse caso, o **DefaultCredentials** utiliza o contexto de segurança no qual a aplicação esteja sendo executado, que é geralmente as credenciais Windows (nome de usuário, senha e domínio) do usuário que esteja executando a aplicação.

## Autenticação Básica HTTP

A autenticação básica HTTP é fornecida pelo IIS. Com a autenticação básica, o IIS solicita aos usuários uma conta e senha válidas do Windows. Essa combinação é passada do cliente para o servidor como texto simples codificado e é utilizada para autenticar o usuário no servidor Web.

Nota: Para proteger a autenticação básica, proteja o canal de comunicação entre o cliente e o servidor (por exemplo, habilitando o Secure Sockets Layer [SSL] no servidor) para garantir que a combinação nome de usuário/senha seja criptografada e não possa ser alterada ou interceptada em trânsito. Proteja também as senhas localizadas no servidor. Entretanto, o SSL proteger a comunicação entre dois pontos finais definidos. Se você solicitar a proteção da comunicação entre mais de dois pontos, utilize a segurança baseada em mensagem.

### Web Services que Utilizam Autenticação Básica

Para uma aplicação cliente interagindo com um serviço da Web configurado para autenticação básica, o cliente pode aceitar credenciais de usuário válidas utilizando uma caixa de diálogo de login e usá-la para autenticação. O código abaixo ilustra como configurar as credenciais do usuário para o proxy do serviço da Web à espera de autenticação básica.

```
CredentialCache cache = new CredentialCache();
cache.Add( new Uri( service.Url ),      // web service URL.
           "Basic",                      // Basic Authentication.
           new NetworkCredential( userName, password, domain ) );
service.Credentials = cache;
```

Nesse caso, **userName** (nome de usuário), **password** (senha), e **domain** (domínio) são aceitos como parte da caixa de diálogo de login.

## Autenticação Digest HTTP

A autenticação digest HTTP oferece os mesmos recursos da autenticação básica HTTP, mas envolve uma forma diferente de transmissão das credenciais de autenticação. As credenciais de autenticação são convertidas em um processo de mão única chamado hashing.

O resultado desse processo é chamado de hash, ou message digest, e não é possível descriptografá-lo utilizando as tecnologias atuais.

A autenticação digest ocorre da seguinte forma:

1. O servidor envia ao navegador determinadas informações que serão utilizadas no processo de autenticação.
2. O navegador adiciona essa informação ao nome de usuário e senha, juntamente com algumas outras informações, e usa hashes. As informações adicionais ajudam a evitar que alguém copie o valor hash e utilize-a novamente.

3. O hash resultante é enviado pela rede ao servidor juntamente com as informações adicionais em texto limpo.
4. O servidor adiciona as informações extras a uma cópia de texto simples que ele tenha da senha do cliente e usa hash em todas as informações.
5. O servidor compara o valor hash recebido com o que acabou de produzir.
6. O acesso é concedido apenas se os dois valores forem idênticos.

As informações complementares são adicionadas à senha antes do hashing de forma que ninguém possa capturar o hash da senha e utilizá-lo para fazer-se passar pelo cliente verdadeiro. Os valores são adicionados para ajudar a identificar o cliente, o computador cliente, o realm, ou o domínio ao qual o cliente pertença. Um time stamp também é adicionado para evitar que um cliente use uma senha que tenha sido revogada.

Pelo fato de a autenticação digest enviar a senha pela rede de forma compilada, ela é nitidamente preferível em relação à autenticação básica, especialmente se você utilizar a autenticação básica sem criptografar o canal de comunicação. Dessa forma, você deve usar a autenticação digest em vez da autenticação básica sempre que possível.

---

**Nota:** Assim como acontece com a autenticação básica, a autenticação digest é concluída somente se o servidor do domínio para o qual seja feita uma solicitação tiver uma cópia em texto simples da senha do usuário solicitante. Pelo fato de o controlador do domínio ter cópias em texto simples de senhas, certifique que esse servidor esteja protegido tanto de ataques físicos quando de rede.

---

### **Autenticação Baseada em Certificado**

Os certificados podem permitir que aplicações de cliente e servidor autenticuem-se mutuamente de modo a estabelecer uma conexão segura utilizando chaves digitais instaladas no computador. A aplicação cliente pode usar certificados cliente para identificar-se para o servidor, da mesma forma que o servidor pode identificar-se para o cliente utilizando um certificado de servidor. Um terceiro, que seja mutuamente confiável, chamado autoridade de certificado, pode confirmar a identidade dos certificados. Certificados cliente podem ser mapeados para contas Microsoft Windows no serviço de diretório Microsoft Active Directory®.

Você pode definir um site, de forma que usuários sem certificados possam logar como convidados; entretanto, os usuários com certificados serão logados como os usuários para aos quais os respectivos certificados mapearem. Você pode, ainda, personalizar o site com base no certificado.

Para autenticar usuários individuais, você utiliza uma técnica chamada mapeamento one-to-one na qual, por exemplo, qualquer certificado que contenha um nome comum de empresa seja mapeado para uma única conta. Se você precisar autenticar todos os usuários de um grupo ou organização em particular, utilize o mapeamento many-to-one no qual, por exemplo, qualquer certificado que contenha um nome comum de empresa seja mapeado para uma única conta.

Na autenticação baseada em certificado, as aplicações cliente utilizam certificados que podem ser autenticados por serviços da Web. Nesse caso, a aplicação cliente assina digitalmente as mensagens SOAP utilizando certificados X.509 para garantir que a mensagem seja de uma fonte confiável e não tenha sido alterada durante o trânsito antes de atingir o serviço da Web designado.

Uma importante consideração da autenticação baseada em certificado é como gerenciar situações nas quais um certificado não deva mais ser válido. Por exemplo, se um empregado utiliza um certificado para ser autenticado e o empregado é demitido, o certificado não deve mais permitir que o usuário acesse recursos. Dessa forma, é importante que sua infra-estrutura de segurança de certificado inclua a administração das listas de revogação de certificado. Essas listas estão presentes no servidor e devem ser verificadas a cada vez que o cliente se conectar a um recurso de rede.

Listas de revogação baseadas no servidor não podem ser verificadas quando um Smart Client entrar em modo offline, assim, há chances de um usuário acessar recursos localmente no cliente que ele não seria capaz de acessar no servidor. Para ajudar a contornar esse problema, você pode optar por ter períodos curtos de tempo de contrato dos certificados de clientes. Esses períodos curtos forçam o cliente a conectar-se regularmente a um servidor de certificado e checar se o certificado não tenha sido revogado antes de renovar o contrato e permitir a conexão de parte do servidor da aplicação.

Para maiores informações, consulte "Sobre Certificados" em

[http://www.microsoft.com/resources/documentation/windowsserv/2003/standard/proddocs/en-us/sec\\_auth\\_certabout.asp](http://www.microsoft.com/resources/documentation/windowsserv/2003/standard/proddocs/en-us/sec_auth_certabout.asp).

### Autenticação baseada em WSE

Você pode assinar as mensagens SOAP de maneira programática para um serviço da Web utilizando a versão 2.0. do Web Services Enhancements. O WSE 2.0 é uma implementação que suporta Web Services padrões como WS-Security, WS-SecureConversation, WS-Trust, WS-Policy, WS-Addressing, WS-Referral e WS-Attachments e Direct Internet Message Encapsulation (DIME). O WSE oferece um modelo de programação para implementar várias especificações por ele suportadas.

Aplicações cliente que utilizam o WSE podem usar um dos métodos **Find** (por exemplo, **FindCertificateByHash** ou **FindCertificateByKeyIdentifier**) na classe **X509CertificateStore** para selecionar um certificado do armazenamento, criar uma assinatura digital utilizando o certificado, adicioná-lo ao cabeçalho SOAP WS-Security, e chamar o serviço da Web. Como alternativa, a aplicação cliente também pode abrir o armazenamento de certificado do usuário atualmente logado, conforme demonstrado no seguinte exemplo de código.

```
x509CertificateStore store;  
  
store = X509CertificateStore.CurrentUserStore(  
    X509CertificateStore.MyStore );  
  
bool open = store.OpenRead();
```

Para maiores informações, consulte "Web Services Enhancements" em

<http://msdn.microsoft.com/webservices/building/wse/default.aspx>.

Para mais detalhes sobre o uso de certificados cliente, consulte "Assinar uma Mensagem SOAP Utilizando um Certificado X.509" na documentação do WSE 2.0.

### Autenticação Personalizada

Em alguns casos, a autenticação padrão oferecida pelo Windows não é apropriada para suas aplicações, e você não precisará desenvolver seu próprio formulário de autenticação. Felizmente, o .NET Framework fornece opções para ajudá-lo a desenvolver uma solução personalizada de autenticação.

O .NET Framework suporta uma implementação de **IIdentity**, chamada **GenericIdentity**. Use o **GenericIdentity** ou criar sua própria classe de identidade personalizada. Desenvolver uma solução personalizada de autenticação pode ser difícil, uma vez que você segue seus próprios passos para certificar-se de que o método seja seguro. Você também pode ter que manter um armazenamento separado para suas identidades.

## Recolher e Validar Credenciais de Usuários

Seja qual for a forma de autenticação utilizada, é necessário coletar credenciais de usuários que possam ser validadas. Para usuários que utilizam a autenticação integrada do Windows, você poderá precisar coletar credenciais existentes, já para uma solução de autenticação personalizada, você precisará recolher credenciais de modo seguro através de sua própria caixa de diálogo de logon.

**Nota:** Não armazene credenciais de usuários em seu código por mais tempo do que seja necessário. De modo particular, não armazene credenciais em variáveis globais que fornecem acesso a elas por meio de métodos ou propriedades publicamente acessíveis, e não as salve em disco.

## Recolher Credenciais de Usuários Atualmente Logados

Se você estiver usando a autenticação integrada do Windows, seus usuários serão logados no início da sessão do Windows. Suas aplicações podem, então, usar essas informações para garantir que elas tenham as credenciais apropriadas para serem executadas.

O código a seguir demonstra as funcionalidades básicas.

```
using System.Security.Principal;

// Get principal of the currently logged in user.

windowsPrincipal wp = new windowsPrincipal(
windowsIdentity.GetCurrent() );

// Display the current user name.

label1.Text = "User:" + wp.Identity.Name;

// Determine if user is part of a windows group.

if( wp.IsInRole( "YourDomain\\YourWindowsGroup" ) )
{
    // Is a member.
}
else
{
    // Is not a member.
}
```

## Recolher Credenciais de Usuários Utilizando uma Caixa de Diálogo de Logon

Se você estiver desenvolvendo sua própria caixa de diálogo de logon para aceitar credenciais de usuários, você precisará tomar algumas medidas para certificar-se de que tenha feito tudo de acordo com os requisitos de segurança da sua organização (como o reforço das políticas fortes de senha e garantir que as senhas expirem periodicamente). Considere as seguintes diretrizes ao desenvolver sua caixa de diálogo de logon:

- **Não confie cegamente em dados de entrada de usuário.** Caso contrário, um usuário com más intenções pode comprometer sua aplicação. Por exemplo, uma aplicação que utilize dados de entrada sem validação de código SQL criado dinamicamente pode estar vulnerável a ataques de injeção SQL.
- **Verifique o tipo, formato ou extensão dos dados de entrada.** Considere o uso de expressões regulares para executar essas verificações. O uso de expressões regulares permite a você verificar o

tipo (por exemplo, string ou integer), formato (por exemplo, cumprimento de requisitos de política de senha como uso de números, caracteres especiais e uma mistura de letras maiúsculas e minúsculas), e extensão (por exemplo, um nome de usuário com um mínimo de 6 e máximo de 25 caracteres).

O exemplo de código a seguir reforça o uso de uma senha que tenha entre 8 e 10 caracteres de extensão juntamente com uma combinação de letras maiúsculas, minúsculas e caracteres numéricos.

```
// validate the user supplied password.
if( !Regex.Match( textBox1.Text,
    @"^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,10}$",
    RegexOptions.None ).Success )
{
    // Invalid email address.
}
```

- Ao desenvolver uma caixa de diálogo, com uma caixa de campo de texto para senha, certifique-se de que a propriedade **PasswordChar** seja definida para um caractere que possa ser exibido quando texto for digitado no controle, conforme demonstrado no exemplo a seguir.

```
// The password character is set to asterisk.
textBox1.PasswordChar = '*';
```

## Diretrizes de Autenticação

Ao desenvolver autenticações para suas aplicações, você deve considerar as seguintes diretrizes:

- Determine quando a autenticação precisa ocorrer durante a interação do usuário com sua aplicação.
- Considere o uso da autenticação integrada do Windows para autenticar os usuários conforme eles se loguem ao cliente e antes que eles possam acessar sua aplicação, dados e qualquer tipo de serviço remoto.
- Se sua aplicação for centralmente implementada e você precisar restringir o acesso apenas a usuários autorizados, autentique os usuários quando a aplicação for executada utilizando um dos mecanismos de autenticação fornecidos pelo IIS.
- Se sua aplicação fornecer funcionalidade sensível ou acesso a dados mantidos localmente, certifique-se de que os usuários estejam apropriadamente autenticados antes de permitir o acesso.
- Se sua aplicação requerer autenticação personalizada, certifique-se de que sua aplicação exija o cumprimento de uma política forte de nome de usuário e senha. Como uma prática geral, você deve requerer um mínimo de 8 caracteres e uma mistura de letras maiúsculas e minúsculas, números e caracteres especiais.
- Solicite autenticação de usuário para acesso a serviços remotos na rede caso eles forneçam acesso a funcionalidades sensíveis ou a dados sensíveis.
- Garanta que as credenciais de usuários não sejam transmitidas de maneira desprotegida pela rede. Algumas formas de autenticação evitam totalmente passar credenciais de usuários pela rede mas, caso elas tenham que ser transmitidas, você deve certificar-se de que elas sejam criptografadas ou enviadas por uma conexão segura.

Para maiores informações, consulte "Autenticação" no "Capítulo 4 Desenvolvimento de Diretrizes para Aplicações Web Seguras" de *Improving Web Application Security: Threats and Countermeasures* em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh04.asp>.

## Autorização

Após os usuários serem autenticados, você pode determinar ao que eles terão acesso no sistema pelo uso da autorização. A autorização é a confirmação de que um usuário autenticado tenha permissão para executar uma operação. A autorização governa os recursos (por exemplo, arquivos e bases de dados) que um usuário pode acessar e as operações (por exemplo, alterar senhas ou excluir arquivos) que podem ser executadas. Os usuários que não forem autenticados (ou seja, usuários anônimos) não serão capazes de ser especificamente autorizados e precisarão ter um conjunto padrão de permissões a eles atribuídas.

Uma série de fatores determina exatamente como você executa a autorização em seu ambiente. Determine se irá gerenciar a autorização baseado nas funcionalidades da aplicação ou nos recursos do sistema. Você precisa decidir se irá executar a autorização no interior do método ou executar verificações ao nível do método.

Determine também onde a informação de usuário solicitada para autorização está armazenada (por exemplo no Active Directory ou em um banco de dados Microsoft SQL Server™). Se você pretende permitir que seus Smart Clients trabalhem em modo offline, você precisa de uma estratégia para a autorização de clientes offline.

O .NET Framework fornece a interface **IPrincipal**, utilizada em conjunto com a interface **IIdentity** para definir as propriedades e métodos para gerenciar o contexto de segurança do código executado. Duas implementações dessa interface também são fornecidas: **WindowsPrincipal** e **GenericPrincipal**. Aplicações cliente que utilizam a autenticação integrada do Windows, usam o **WindowsPrincipal**, ao passo que aplicações cliente que utilizam a autenticação personalizada, usam o **GenericPrincipal**.

### Tipos de Autorização

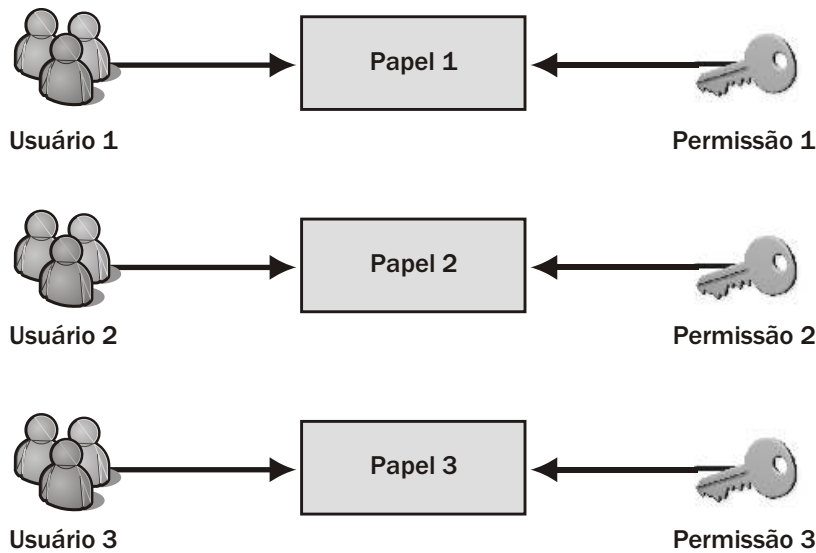
Dois métodos de autorização são comumente utilizados no sistema operacional Windows: a autorização baseada em recursos e a autorização baseada em funções. A autorização baseada em recursos confia nas listas de controle de acesso (ACLs), enquanto que a autorização baseada em funções executa a autorização baseada nas funções dos usuários.

#### Autorização Baseada em Recursos

Para a autorização baseada em recursos, anexe listas discricionárias de controle de acesso a objetos protegíveis. O sistema, então, passará a tomar decisões relativas ao acesso comparando os membros de grupo aos conteúdos das ACL's para determinar se o usuário tem o acesso solicitado. O modelo ACL é o ideal para muitos tipos de aplicação. Entretanto, ele não é apropriado para todas as situações. Por exemplo, você pode precisar tomar decisões relativas ao acesso baseadas em lógica business ou em objetos não-persistentes criados quando necessários.

#### Autorização Baseada em Funções

A autorização baseada em funções permite a você associar usuários e grupos com as permissões que eles precisam para executar seus trabalhos. Quando um usuário ou grupo é adicionado a um papel, o usuário ou grupo herda, automaticamente, as várias permissões de segurança. Essas podem ser permissões para executar ações ou para acessar vários recursos. A Figura 5.1 mostra a relação entre as funções e permissões em uma autorização baseada em funções.

**Figura 5.1**

*Autorização baseada em funções*

Nos sistemas operacionais Microsoft Windows 2000 Server Service Pack 4 (SP4) e Windows Server™ 2003, a autorização baseada em funções é geralmente administrada utilizando o Gerenciador de Autorizações. O Gerenciador de Autorizações é um conjunto de interfaces run-time baseadas em COM, juntamente com um snap-in Microsoft Management Console (MMC) para configuração. Os desenvolvedores podem usar o Gerenciador de Autorizações para garantir que as aplicações possam gerenciar e verificar solicitações de cliente para executar operações da aplicação, e os administradores da aplicação podem usá-lo para gerenciar funções e permissões de usuários. Com o Gerenciador de Autorizações, agregue operações de baixo nível em grupos chamados Tasks (Tarefas) e gerenciar a autorização nesse nível. Ele também permite a você executar lógica de autorização personalizada antes e depois da autorização.

Uma vantagem significativa do Gerenciador de Autorizações é a de que ele posteriormente abstrai o armazenamento de autorizações da aplicação que requer autorização, significando que os desenvolvedores sempre poderão comunicar-se com o Gerenciador de Autorizações, independentemente se o armazenamento estiver no Active Directory ou baseado em arquivo.

## **Adição de Capacidades de Autorização em sua Aplicação**

O .NET Framework fornece uma série de opções para adicionar capacidades de autorização em sua aplicação, incluindo:

- **Executar demandas declarativas utilizando o `PrincipalPermissionAttribute`**
- **Executar demandas imperativas utilizando o objeto `PrincipalPermission`**
- **Executar verificações de funções utilizando o método `IsInRole`**
- **Executar verificações de funções para autenticação personalizada**

### **Executar Demandas Declarativas Utilizando o `PrincipalPermissionAttribute`**

Você pode posicionar demandas ao nível da classe, ou ao nível do membro em métodos, propriedades ou eventos individuais. Se você posicionar uma demanda declarativa ao nível da classe e do membro, a demanda declarativa ao nível do membro substitui a demanda ao nível da classe.



O seguinte exemplo de código exibe uma demanda declarativa para o objeto **PrincipalPermission**.

```
// Declarative example.
[PrincipalPermissionAttribute( SecurityAction.Demand, Role="Teller" )]
void SomeTellerOnlyMethod()
{
}
}
```

### Executar Demandas Imperativas Utilizando o Objeto PrincipalPermission

Você pode executar demandas imperativas chamando, de maneira programática, o método Demand do objeto PrincipalPermission, conforme demonstrado no seguinte exemplo de código.

```
// Programmatic example.
public SomeMethod()
{
    PrincipalPermission permCheck = new PrincipalPermission( null,
        "Teller" );
    permCheck.Demand();
    // Only Tellers can execute the following code.
    // Non members of the Teller role result in a security exception.
    . . .
}
```

Uma vantagem de chamar o método de maneira programática é que você pode determinar se o principal está em mais de um papel. O .NET Framework não permite a você fazer isso de maneira declarada. O seguinte código de exemplo demonstra como executar a verificação.

```
// Using PrincipalPermission.
PrincipalPermission permCheckTellers = new PrincipalPermission( null,
    "Teller" );
permCheckTellers.Demand();

PrincipalPermission permCheckMgr = new PrincipalPermission( null,
    "Manager" );
permCheckMgr.Demand();
```

### Executar Verificações de Funções Utilizando o Método IsInRole

Você pode optar por acessar os valores do objeto principal diretamente e executar as verificações sem um objeto **PrincipalPermission**. Nesse caso, você pode ler os valores do thread principal atual ou usar o método **IsInRole** para executar a autorização, conforme demonstrado no seguinte exemplo de código.

```
// Using IsInRole.
if ( Thread.CurrentPrincipal.IsInRole( "Teller" ) &&
    Thread.CurrentPrincipal.IsInRole( "Manager" ) )
```

```
{
    // Perform privileged operation.
}
```

### Executar Verificações de Funções para Autenticação Personalizada

Se sua aplicação não for baseada em Windows, você pode preencher, através de código, o objeto **GenericPrincipal** com um conjunto de funções obtidos de um armazenamento de dados de autenticação personalizada como um banco de dados SQL Server, conforme demonstrado no seguinte exemplo de código.

```
GenericIdentity userIdentity = new GenericIdentity( "Bob" );
// Typically role names would be retrieved from a custom data store.
string[] roles = new String[]{"Manager", "Teller" };
GenericPrincipal userPrincipal = new GenericPrincipal( userIdentity,
    roles );
if ( userPrincipal.IsInRole( "Teller" ) )
{
    // Perform privileged operation.
}
```

### Diretrizes de Autorização

A autorização é um fator crítico para controlar o acesso de usuários às funcionalidades da aplicação e recursos acessados. Uma autorização imprópria ou fraca leva à publicação da informação e manipulação dos dados. Considere as seguintes diretrizes de autorização:

- **Use gatekeepers múltiplos onde for possível fazer as verificações de autorização ao acessar recursos ou executar operações.** O uso de verificações no cliente, combinado com verificações no servidor, fornece um alto nível de defesa para evitar um ataque de usuário com más intenções que consiga passar por um dos gatekeepers. Os gatekeepers comuns no servidor incluem permissões Web IIS, permissões NTFS do sistema para arquivos, autorização de arquivo de serviço da Web (que se aplica somente com autenticação Windows) e demandas principais de permissão.
- **Autorize o acesso aos recursos do sistema utilizando o contexto de segurança do usuário.** Você pode usar a autorização baseada em funções, para autorizar o acesso baseado na identidade do usuário e filiação ao papel. A autenticação integrada do Windows com ACL's do Windows em recursos protegidos (como arquivos ou o registro) determina se o chamado tem permissão de acesso para o recurso. Para módulos, autorize código de chamada baseado em evidência, como um nome ou local de módulo.
- **Garanta que as funções sejam definidas com granularidade suficiente para separar os privilégios de maneira adequada.** Evite conceder privilégios elevados apenas para satisfazer os requisitos de certos usuários; em vez disso, considere a opção de adicionar novas funções para estar de acordo com tais requisitos.
- **Quando possível, use demandas declarativas em vez de demandas imperativas.** As demandas declarativas fornecem ou negam acesso para todas as funcionalidades do método. Elas também trabalham muito melhor com ferramentas de segurança e ajudam na questão de auditorias de segurança, uma vez que as ferramentas são capazes de acessar essas demandas examinando a aplicação.

- **Se você precisar determinar se o principal está em mais de um papel, considere o uso de verificações imperativas utilizando o método `IsInRole`.** O .NET Framework versão 1.1 não permite e checa para ser programado de maneira declarativa; entretanto, elas podem ser executadas de maneira programática no interior do método, conforme demonstrado no seguinte exemplo de código.

```
// Checking for multiple roles.
if ( Thread.CurrentPrincipal.IsInRole( "Teller" ) &&
    Thread.CurrentPrincipal.IsInRole( "Manager" ) )
{
    // Perform privileged operation.
}
```

- **Use a segurança de acesso ao código para autorizar acesso do código de chamada para recursos privilegiados ou operações, baseados em evidência, como um nome forte ou local de módulo.** Para maiores informações, consulte "Segurança de Acesso ao Código", mais adiante nesse capítulo.

## Autorização de Funcionalidade quando o Cliente Está Offline

Quando os usuários estão conectados à rede, eles podem ser autorizados diretamente contra um armazenamento de autorizações mas, quando não estão, eles podem ainda necessitar de autorização.

Qualquer forma de autorização é tão forte quanto o mecanismo de autenticação utilizado.

Se você permitir autenticação anônima, você deve tomar muito cuidado em relação a quais funcionalidades são permitidas ao acesso dos usuários e, de maneira geral, não deve autorizar que os usuários acessem recursos do sistema.

Se você estiver autenticando usuários para utilizar uma aplicação, você pode deixar que o Windows aja como o único gatekeeper para determinar quais recursos estejam disponíveis para o perfil do usuário logado. Nesse caso, ao usuário é sempre permitido o acesso aos recursos locais do sistema. Você pode optar por criar diferentes versões da mesma aplicação para funções diferentes.

Quando o usuário está conectado à rede, ele tem a permissão de instalar apenas a versão da aplicação feita para seu papel. A seguir, quando o usuário executar a aplicação offline, a funcionalidade correta é automaticamente fornecida sem a necessidade de a aplicação estar conectada.

## O Bloco de Aplicação Autorização e Perfil

A Microsoft oferece um bloco de aplicação que fornece a infra-estrutura para simplificar a inclusão da funcionalidade de autorização em sua aplicação.

O Bloco de Aplicação Autorização e Perfil fornece uma infra-estrutura para autorização baseada em funções e acesso à informações de perfil. O bloco permite a você:

- Autorizar um usuário de uma aplicação ou sistema.
- Usar múltiplos fornecedores de armazenamento de autorização.
- Plug-in business rules para validação de ação.
- Mapear identidades múltiplas para um único usuário, estendendo a idéia de uma identidade para incluir métodos de autenticação.
- Acessar informações de perfil que possam ser armazenadas em múltiplos armazenamentos de perfil.

Para mais informações, consulte Authorization and Profile Application Block em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/authpro.asp>.

## Validação de Entrada

Aplicações com fraca validação de entrada podem ser comprometidas por dados de entrada maliciosos em um ataque. A opção de validar os dados de entrada do usuário é uma das principais formas de defesa para sua aplicação. Considere as seguintes diretrizes de validação de dados de entrada para sua aplicação Smart Client:

- Certifique-se de que sua aplicação Smart Client valide todos os dados de entrada antes de processar ou passá-los para recursos e módulos downstream.
- Execute uma validação completa dos dados de entrada do usuário se estiver passando-os para um API não-gerenciado. Isso evitará o sobrecarregamento do buffer. Você deve limitar os dados de entrada de usuário que são passados para API's não-gerenciados.
- Sempre valide os dados obtidos de todas as fontes externas, como sites e serviços da Web.
- Nunca confie na validação de dados por parte do cliente e que seja passada para seu serviço ou aplicação Web. Valide os dados no cliente e, a seguir, valide-os novamente no servidor a fim de evitar dados maliciosos de entrada que possam passar ilesos pela validação por parte do cliente.
- Nunca permita que os usuários entrem queries SQL diretamente. Sempre forneça queries pré-estabelecidas ou padronizadas que são completamente revisadas em busca de problemas de segurança. Permitir que os usuários entrem queries SQL diretamente levanta a possibilidade de ataques SQL de injeção.
- Restrinja e valide dados de entrada de usuário para valores ou padrões corretos conhecidos, e não para dados incorretos. É mais fácil verificar uma lista limitada de valores conhecidos do que verificar uma lista infinita de tipos de dados desconhecidos e maliciosos. Você pode tanto rejeitar o dado ruim quanto limpá-lo (ou seja, retirar caracteres potencialmente não-seguros) antes de utilizá-lo.
- Restrinja os dados de entrada validando-os de acordo com o tipo, tamanho, formato e extensão. Uma forma de fazer isso é utilizar expressões regulares (disponível a partir do namespace **System.Text.RegularExpressions**) para validar dados de entrada de usuário.
- Rejeite dados ruins e, em seguir, limpe os dados. Se sua aplicação precisar aceitar alguns dados de entrada de usuário em um formulário livre (por exemplo, comentários em uma caixa de texto, limpe os dados de entrada conforme demonstrado no exemplo abaixo.

```
private string SanitizeInput( string input )
{
    // Example list of characters to remove from input.
    Regex badCharReplace = new Regex( @"(<>"'"%";()&)]" );
    string goodChars = badCharReplace.Replace( input, "" );
    return goodChars;
}
```

- Considere a opção de centralizar suas rotinas de validação para reduzir o esforço de desenvolvimento e ajudar a manutenção futura.

Para maiores informações, consulte "Validação de Dados de Entrada" no "Capítulo 4 Diretrizes de Design para Aplicações Web Seguras" de Improving Web Application Security: Threats and Countermeasures em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh04.asp>.

## Manipulação de Dados Sensíveis

Se você estiver acostumado a desenvolver aplicações Web, você saberá a importância de proteger dados armazenados e dados em trânsito. Os dados que você armazena em um servidor Web são tipicamente gravados em um local fisicamente seguro que já esteja bem protegido para evitar ataques. Em aplicações Smart Client, você também precisa considerar os dados que residam no cliente. Caso esses dados sejam sensíveis, é importante manipulá-los de modo a garantir sua segurança. Para proteger dados em trânsito, você pode proteger a camada de transporte utilizando SSL e proteger o conteúdo da mensagem as ferramentas de criptografia de WS-Security ou Message Queuing.

Apenas os dados para os quais o usuário tem acesso autorizado devem estar disponíveis para a aplicação cliente. Se a aplicação cliente puder ser usada por mais de uma pessoa em um mesmo computador, os dados associados a cada usuário individual devem ser considerados dados sensíveis, e os devidos passos devem ser dados para garantir que apenas usuários autorizados possam acessá-los.

Os dados sensíveis incluem quaisquer dados que um atacante possa considerar úteis para acessar ou modificar, seja por que a informação é confidencial, ou por que ela possa ajudar em um ataque. Os dados sensíveis podem ser dados que o servidor fornece ao cliente, mas também podem incluir arquivos de configuração de aplicação, bases de dados locais, ou informações de registro.

Em geral, tente garantir que os dados sensíveis não sejam armazenados em cache localmente. Contudo, no caso de uma aplicação Smart Client, você pode precisar armazenar esses dados em cache (por exemplo, para permitir uma operação ocasionalmente conectada salvando os dados em um armazenamento local para uso posterior).

---

**Nota:** Em alguns casos, os dados sensíveis podem ser enviados para o disco como um resultado de paging da memória. Assim, você deve, também, considerar os dados presentes na memória ao determinar quais dados precisam ser criptografados.

---

## Determinar Quais Dados Armazenar no Cliente

Por definição, os usuários e, conseqüentemente, os atacantes em potencial, têm acesso físico aos clientes. Dado tempo suficiente, os atacantes sempre são capazes de obter acesso administrativo suficiente para acessar praticamente qualquer tipo de dado, de forma que você deve considerar com ponderação quais dados manter no cliente. Como regra geral, você deve tomar as decisões de autorização no servidor, de modo que somente os dados que você passar do servidor para o cliente sejam os dados aos quais o usuário tenha acesso. Além de aumentar o desempenho, tomar as decisões de autorização no servidor também garante que os dados não estejam disponíveis no cliente para um atacante em potencial poder acessar.

Você jamais deve armazenar dados sensíveis em arquivos baseados em texto e deve sempre criptografar os dados para que eles possam ser facilmente acessados somente por usuários autorizados. Você deve evitar o uso de arquivos de configuração baseados em texto para armazenar informações sensíveis de segurança, como senhas ou cadeias de conexão de banco de dados. Se essa informação precisar ser armazenada localmente, você deverá criptografar a informação, armazená-la em um arquivo ou chave de registro e, a seguir, restringir o acesso para aquele objeto com um DACL. Quaisquer dados pessoais ao usuário logado também devem ser mantidos de maneira confidencial e segura, principalmente se o computador for compartilhado entre usuários.

Em muitos casos, mais dados são armazenados no cliente se a aplicação precisar ser executada em modo offline. Entretanto, você deve determinar se todos os dados são necessários em modo offline ou se você deseja restringir a execução de certas ações por parte do usuário quando em modo offline, de modo que não tenha que armazenar dados sensíveis localmente.

Em alguns casos, se os dados são confidenciais e podem ser inseridos pelo usuário em demanda, você pode optar por não armazená-los localmente no cliente e, em vez disso, obtê-los do usuário conforme necessário.

Se sua aplicação precisa armazenar dados sensíveis localmente, você deve evitar o uso de armazenamento removível (como disquetes, zip disks ou equipamentos USB de armazenamento) ou de armazenamento portátil externo para armazenar dados sensíveis. Contudo, dados específicos do usuário podem ser armazenados em mídias removíveis quando você puder ter certeza de que a mídia removível pertença ao usuário (por exemplo, ao utilizar um certificado ou um smart card). Assim, dados específicos do usuário podem ser mantidos em um local seguro que viaje junto ao usuário, de modo que usuários em roaming possam acessar a aplicação e seus dados sem deixar esses dados no computador local.

---

**Nota:** Ao considerar quais dados sensíveis deverão ser armazenados no cliente, você deve certificar-se de que, ao armazenar informações sobre seus empregados ou consumidores, você não esteja violando regras de privacidade. Essas leis diferem de acordo com o país, assim, você deve familiarizar-se com as regras nos países nos quais sua aplicação seja utilizada.

---

## Técnicas para Proteção de Dados Sensíveis

Para os dados que você precisa armazenar no cliente, há uma série de medidas adicionais para evitar o acesso não-autorizado, que incluem:

- **Certificar-se de que apenas usuários autorizados possam acessar os dados.**
- **Considerar o uso de EFS para criptografar arquivos.**
- **Considerar o uso de DPAPI para evitar questões de gerenciamento de chave.**
- **Considerar o armazenamento de valores hash em vez de texto simples.**
- **Considerar o armazenamento isolado para aplicações parcialmente confiáveis.**
- **Proteger suas chaves particulares.**

### Certificar-se de que Apenas Usuários Autorizados Possam Acessar os Dados

Seus dados sempre precisarão ser protegidos para ajudá-lo a certificar-se de que apenas usuários autorizados possam acessá-los. Dependendo da natureza de sua aplicação e de quão transitável forem os dados, você pode optar por utilizar a segurança baseada em recursos ou a segurança baseada em funções para proteger seus dados. Para maiores informações, consulte "Diretrizes de Autorização" previamente neste capítulo.

### Considerar o Uso de EFS para Criptografar Arquivos

Uma opção para certificar-se de que os arquivos estejam sendo mantidos de maneira segura em Smart Clients é usar o Encrypting File System (EFS) para criptografar arquivos de dados sensíveis. Essa solução não é particularmente escalável; entretanto, ela pode ser útil para arquivos específicos, e pode ser útil para situações nas quais você esteja armazenando dados em cache localmente no cliente (por exemplo, para habilitar Smart Clients ocasionalmente conectados).

### Considerar o Uso de DPAPI para Evitar Questões de Gerenciamento de Chave

O Windows 2000 e versões posteriores do sistema operacional Windows fornecem o Win32® Data Protection API (DPAPI) para criptografar e descriptografar dados. Ele é parte do Cryptography API (Crypto API) e é implementado no crypt32.dll, consistindo em dois métodos: **CryptProtectData** e **CryptUnprotectData**.

O DPAPI é especialmente útil porque pode eliminar o problema de gerenciamento de chave exposto a aplicações que utilizam a criptografia. Mesmo com a garantia da criptografia de que os dados estão seguros, você deve tomar mais algumas medidas para garantir a segurança da chave. Para derivar a chave de criptografia, o DPAPI utiliza a senha da conta de usuário associada ao código que chama as funções DPAPI. Como resultado, o sistema operacional (e não a aplicação) gerencia a chave.

O DPAPI trabalha tanto com o armazenamento da máquina quanto com o armazenamento de usuário. O armazenamento de usuário é carregado automaticamente com base no perfil do usuário logado. Suas aplicações cliente irão utilizar, na maioria das vezes, o DPAPI com o armazenamento de usuário, a menos que haja a necessidade de armazenar segredos que sejam comuns a todos os usuários que possam logar-se ao computador.

As chaves utilizadas pelo DPAPI para criptografar e descriptografar dados sensíveis são específicas de um computador. Uma chave diferente é gerada para cada computador, o que evita que um servidor possam acessar dados criptografados por outro servidor.

O DPAPI não-gerenciado requer módulos para ter total confiança. As aplicações que possuem módulos totalmente e parcialmente confiáveis podem isolar os códigos com altos privilégios e habilitá-lo para serem chamados a partir de um código parcialmente confiável. Para maiores informações, consulte "Como Criar uma Permissão de Criptografia Personalizada" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod115.asp>.

### **Considerar o Armazenamento de Valores Hash em vez de Texto Simples**

Às vezes, os dados são armazenados de uma forma que possam ser utilizados para validar dados de entrada de usuário (por exemplo, uma combinação de nome de usuário e senha). Nesses casos, em vez de armazenar os dados em texto simples, você pode armazenar um hash criptográfico dos dados. A seguir, quando os dados de entrada de usuários forem feitos, os dados também poderão ser hashed, e ambos podem ser comparados. Armazenar o hash reduz o risco do segredo ser descoberto, uma vez que é computacionalmente impossível deduzir os dados originais a partir do hash ou gerar um hash idêntico a partir de outros dados.

### **Considerar o Armazenamento Isolado para Aplicações Parcialmente Confiáveis**

O armazenamento isolado permite a sua aplicação salvar os dados em um compartimento único de dados que esteja associado com algum aspecto da identidade do código, como seu site Web, publisher ou assinatura. O compartimento de dados é uma abstração, não um local específico de armazenamento; ele consiste em um ou mais arquivos de armazenamento isolados, chamados stores, que contêm os locais reais de armazenamento dos dados.

O armazenamento isolado pode ser especialmente útil para aplicações parcialmente confiáveis que necessitam armazenar dados de estado específicos para usuários e módulos particulares. As aplicações parcialmente confiáveis não têm acesso direto ao sistema de arquivo para persistir o estado, a menos que elas tenham sido agraciadas explicitamente com a permissão para fazerem isso através de uma alteração na política de segurança.

Os dados gravados separadamente são armazenados, isolados e protegidos de outras aplicações parcialmente confiáveis, ainda que não estejam protegidos de código totalmente confiável ou de outros usuários que tenham acesso ao computador cliente. Para proteger os dados nesses cenários, você deve utilizar a criptografia de dados e segurança de sistema de arquivo por meio da utilização de DACLs. Para maiores informações, consulte "Introdução ao Armazenamento Isolado" no .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconIntroductionToIsolatedStorage.asp>.

### **Proteger Chaves Particulares**

Chaves particulares não protegidas são suscetíveis a uma vasta gama de ataques por parte de usuários ou códigos maliciosos. Chaves particulares utilizadas para assinar módulos não devem ser deixadas em

locais desprotegidos, como computadores de desenvolvedores ou em ambientes abertamente compartilhados.

Chaves particulares roubadas podem ser utilizadas por um atacante para assinar um código malicioso com seu nome forte. Você deve considerar a opção de proteger suas chaves particulares com uma autoridade central de segurança designada para esse fim dentro de sua organização. Mantenha suas chaves particulares em um computador isolado e fisicamente protegido, transferindo as chaves, quando necessário, com mídia portátil.

Para maiores informações sobre como guardar segredos de maneira efetiva, consulte *Writing Secure Code*, Second Edition, de Michael Howard e David LeBlanc.

## Segurança do Acesso ao Código

A segurança do acesso ao código é a tecnologia .NET Framework que aplica princípios de autorização e autenticação ao código em vez de usuários. A segurança do acesso ao código pode ser um mecanismo poderoso para garantir que somente o código que você deseje que seja executado seja executado pelo usuário.

Todos os códigos gerenciados estão sujeitos à segurança do acesso ao código. Quando um módulo é carregado, ele ganha um conjunto de permissões de acesso ao código que determinam quais são os recursos que podem ser acessados e quais os tipos de operações privilegiadas que ele pode executar. O sistema de segurança .NET Framework utiliza evidências para autenticar (identificar) o código de modo a conceder essas permissões.

---

**Nota:** Um módulo é a unidade de configuração e confiança para segurança do acesso ao código. Todos os códigos no mesmo módulo recebem as mesmas permissões e são, dessa forma, igualmente confiáveis.

---

A segurança do acesso ao código consiste nos seguintes elementos:

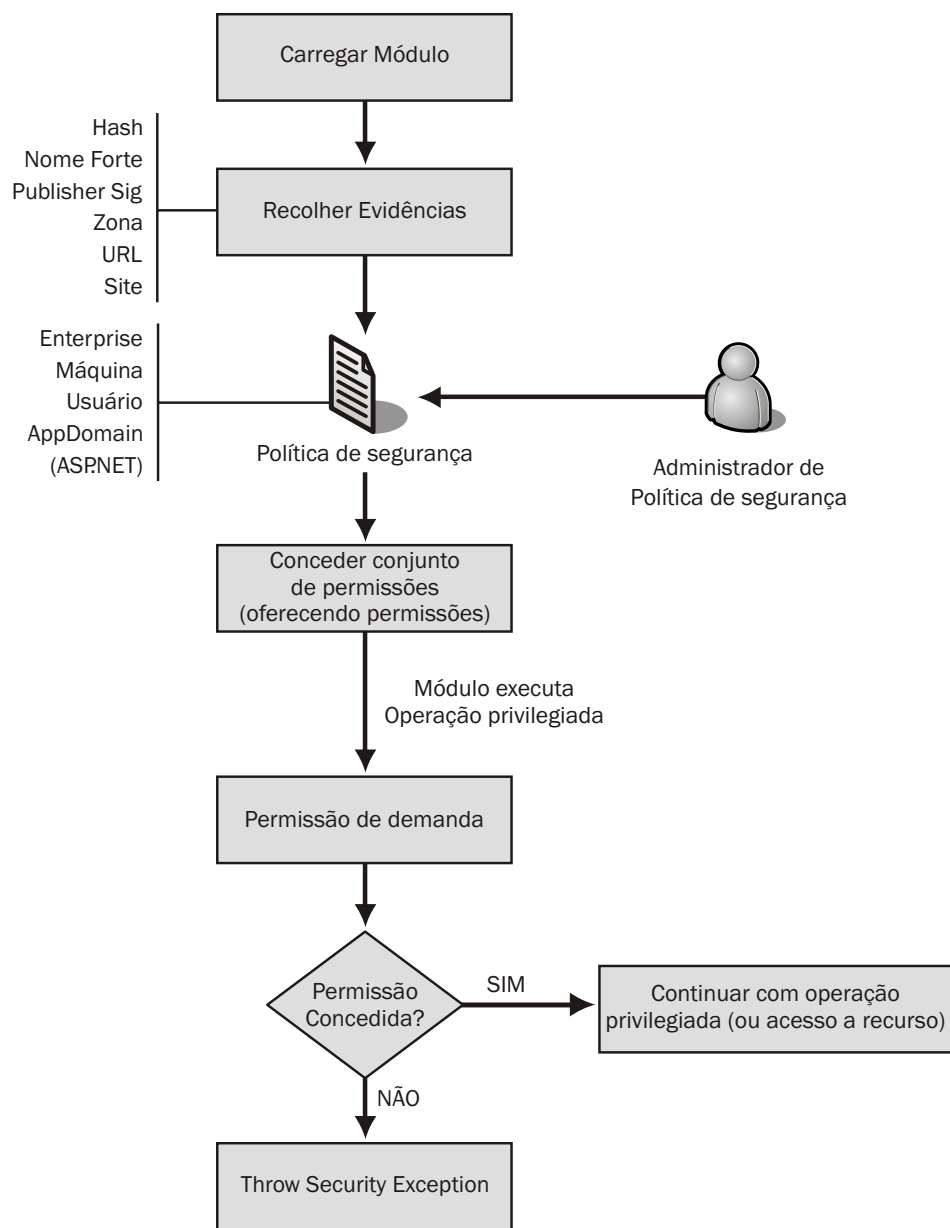
- **Permissões.** As permissões representam os direitos do código acessar um recurso protegido ou executar uma operação privilegiada. O Microsoft .NET Framework fornece permissões de acesso ao código e permissões de identidade do código. As permissões de acesso ao código englobam a habilidade de acessar um determinado recurso ou executar uma operação privilegiada em particular. Por exemplo, o **FileIOPermission** é necessário antes de a aplicação poder executar quaisquer operações I/O de arquivo. As permissões de identidade do código são utilizadas para restringir o acesso ao código, baseadas no aspecto da identidade do código chamado, como seu nome forte.
- **Conjuntos de permissão.** O .NET Framework define um número de conjuntos de permissões, que representam um grupo de permissões comumente atribuídas como um todo. Por exemplo, o .NET Framework define o conjunto de permissão **FullTrust**, que atribui todas as permissões para códigos totalmente confiáveis, e o conjunto de permissão **LocalIntranet**, que atribui um número muito limitado de permissões.
- **Evidence.** A evidência é utilizada pelo sistema de segurança .NET Framework para identificar módulos. A política de acesso ao código utiliza evidências para ajudar a conceder as permissões corretas para o módulo correto. A evidência pode estar relacionada ao local (por exemplo, URL, site, diretório de aplicação ou zona) ou relacionada ao autor (por exemplo, nome forte, publisher, ou hash).
- **Política.** A política de segurança do acesso ao código é configurada por administradores e determina as permissões concedidas aos módulos. A política pode ser estabelecida nos níveis de enterprise, máquina, usuário e domínio de aplicação. Cada política é definida em um arquivo de configuração em XML.
- **Grupos de código.** Cada arquivo de política contém uma coleção hierárquica de grupos de código. Os grupos de código são utilizados para atribuir permissões a módulos. Um grupo de código consiste em uma condição de filiação (baseada em evidências) e em um conjunto de permissões. O .NET Framework define um número de grupos de código padrão, como Internet, Intranet local, Restrito e Zonas confiáveis.



Para maiores informações sobre segurança do acesso ao código, consulte os seguintes capítulos de *Improving Web Application Security: Threats and countermeasures*: "Chapter 7 – Building Secure Montagens" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh07.asp> e "Chapter 8 Code Access Security in Practice" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh08.asp>.

## Resolução de Permissão de Segurança do Acesso ao Código

A segurança do acesso ao código utiliza os passos realçados na Figura 5.2 para determinar quais permissões são atribuídas para um módulo.



**Figura 5.2**

Determinando quais permissões serão atribuídas para um módulo

Os passos a seguir descrevem o procedimento com maiores detalhes:

1. Um módulo é carregado, e evidências são recolhidas e apresentadas ao host.
2. A evidência é avaliada de acordo com a política de segurança para o ambiente hosting.
3. O resultado dessa avaliação é um conjunto de permissões concedidas ao módulo. Essas permissões definem o que o módulo pode ou não fazer nesse ambiente.
4. Quando o módulo pede para executar uma operação privilegiada, as demandas daquela operação são comparadas com as permissões do módulo. Se o módulo tiver permissão, ao código é permitido executar a operação, caso contrário, uma exceção de segurança é lançada.

## Desenvolvimento para Segurança do Acesso ao Código

As permissões atribuídas a seu código dependem da evidência associada com seu código e a política de segurança vigente no computador cliente. Para garantir a segurança de sua aplicação enquanto mantém sua funcionalidade, você precisa considerar cuidadosamente as permissões requeridas pela sua aplicação e a forma pela qual essas permissões são concedidas.

As aplicações às quais são concedidas todas as permissões (aquelas aplicações definidas no conjunto de permissões FullTrust) são conhecidas como aplicações totalmente confiáveis. As aplicações às quais não são concedidas todas as permissões são conhecidas como aplicações parcialmente confiáveis. Em teoria, é geralmente preferível desenvolver suas aplicações para serem parcialmente confiáveis.

Entretanto, aplicações Smart Client precisam, com frequência, executar uma série de operações que as aplicações parcialmente confiáveis não podem executar por padrão. Essas operações incluem:

- Acessar servidores que não seja aquele a partir do qual a aplicação está sendo executada ou acessar servidores que utilizem um protocolo diferente
- Acessar o sistema de arquivo local
- Acessar e interagir com aplicações Microsoft Office locais
- Acessar e interagir com códigos não-gerenciados, como objetos COM

Se sua Smart Client precisa executar esses tipos de operações, você deve considerar a opção de fazer, dela, uma aplicação totalmente confiável ou conceder a ela as permissões adicionais específicas que ela necessita para operar de maneira adequada.

---

Nota: As aplicações implementadas utilizando a implementação no-touch são, por padrão, automaticamente parcialmente confiáveis. Se sua Smart Client precisa executar operações adicionais que não sejam executadas por aplicações parcialmente confiáveis, você precisa implementar uma nova política de segurança ou utilizar um método alternativo para implementar a aplicação.

---

Desenvolver e construir aplicações parcialmente confiáveis pode ser uma tarefa desafiadora, mas o fato de considerar com cuidado e restringir as permissões que são concedidas para sua aplicação ajuda a garantir que ela não possa executar ações inapropriadas ou acessar recursos que não sejam explicitamente solicitados.

Todo código precisa ter a permissão para ser executado antes da execução, mas o código que acessa recursos protegidos ou executa outras operações sensíveis à segurança (como chamar código não gerenciado ou acessar o sistema de arquivo local) deve ter permissões adicionais concedidas pelo .NET Framework para ser capaz de funcionar. Esse tipo de código é conhecido como código privilegiado. De maneira similar, o código não-privilegiado não precisa acessar recursos sensíveis e necessita somente de permissão para ser executado. Ao desenvolver e construir sua aplicação e seus módulos, você deve identificar e documentar ambos os tipos de código. Ao fazer isso, você estará determinando as permissões requeridas por seu código.

Você também deve examinar com cuidado qual evidência é utilizada pelo .NET Framework para atribuir permissões para seu código. A evidência baseada no local da aplicação (por exemplo, um compartilhamento de arquivo ou servidor Web) deve ser considerada somente se o local central for seguro. De maneira similar, as aplicações cujas evidências forem baseadas em uma chave comum utilizada para assinar todos os códigos (por exemplo, pelo departamento de TI de uma organização) devem ser utilizadas somente quando a chave for segura. Entretanto, é geralmente mais seguro confiar em uma evidência de nome forte do que em uma evidência baseada em local, como um endereço de servidor Web.

## Desenvolver Aplicações Parcialmente Confiáveis

Use as seguintes diretrizes ao desenvolver aplicações parcialmente confiáveis:

- **Conhecer os cenários de implementação da sua aplicação**
- **Evitar demandas de permissões que gerem exceções**
- **Utilizar o padrão Demanda/Assert para chamadores parcialmente confiáveis**
- **Considerar o uso de nomes fortes para seus módulos**
- **Evitar conceder total confiança para zonas restritas**

### Conhecer os Cenários de Implementação de sua Aplicação

Tenha uma compreensão clara dos cenários de implementação de sua aplicação durante o desenvolvimento, uma vez que o local no qual sua aplicação é implementada tem um efeito significativo nas permissões concedidas à aplicação por padrão. As funcionalidades da aplicação, como exibição de caixa de diálogo (por exemplo, utilizando um `SaveFileDialog`) ou acesso a recursos do sistema, podem ser restringidas com base no local de implementação.

De modo particular, as permissões concedidas a sua aplicação dependem da zona na qual ela esteja localizada (por exemplo, a zona de Internet, a zona da Intranet local ou zona Confiável). O usuário tem algum controle sobre a filiação à aplicação na zona Confiável, mas você não deve confiar no usuário para inserir sua aplicação nessa zona de modo a garantir uma correta funcionalidade. Desenvolva sua aplicação para falhar caso permissões insuficientes sejam concedidas à ela em tempo de execução.

Se um usuário tentar executar uma ação e a aplicação não tiver permissões suficientes para executar a ação, a tentativa pode resultar em uma demanda falha de permissão o que, por sua vez, gerará uma exceção de segurança. Sua aplicação precisa manejar essas exceções, ou ela irá falhar. Você deve certificar-se de que tais falhas sejam bem manipuladas e deve conceder informações suficientes ao usuário para endereçar o problema sem revelar informações inapropriadas ou sensíveis e relacionadas à segurança.

---

**Nota:** As aplicações implementadas com os recursos de implementação ClickOnce do .NET Framework versão 2.0 terão permissões específicas concedidas de acordo com suas manifestações de implementação. As permissões concedidas serão fixadas quando a aplicação for implementada, e a inserção do local da aplicação na zona Confiável não afetará as permissões que forem concedidas.

---

### Evitar Demandas de Permissão que Gerem Exceções

Determine a permissão necessária para cada uma das funcionalidades de suas aplicações para que possam ser executadas de maneira adequada sem gerar exceções. Considere o seguinte:

- **Desenvolva uma alternativa para evitar a demanda de permissão que possa causar exceções.** Por exemplo, para aplicações baseadas em intranet, em vez de a aplicação abrir automaticamente e ler um arquivo do disco rígido, você pode usar o `OpenFileDialog` para exibir uma caixa de diálogo que instrua o usuário sobre como selecionar o arquivo.

- **Verifique as permissões para que possam lidar bem com as exceções (de maneira especial, `SecurityException`).** Em seu código, crie uma instância de uma classe de permissão específica para o recurso que você esteja tentando acessar e verifique se há as permissões necessárias antes de acessar o recurso. Por exemplo você pode usar a classe `FileDialogPermission` e o método estático `SecurityManager.IsGranted` para verificar as permissões quando tiver uma caixa de diálogo sendo exibida utilizando `OpenFileDialog` ou `SaveFileDialog`, conforme segue.

```
FileDialogPermission fileDialogPermission = new
    FileDialogPermission( FileDialogPermissionAccess.Save );
if ( !SecurityManager.IsGranted( fileDialogPermission ) )
{
    // Not allowed to save file.
}
```

---

**Nota:** `IsGranted` não garante que uma operação tenha êxito porque ela não atravessa a pilha para determinar se todos os códigos upstream tenham as permissões necessárias.

---

- Considere a opção de fazer protótipos e testar o cenário de sua aplicação para várias zonas de implementação:
  - Se sua aplicação for desenvolvida para ser executada a partir de um compartilhamento de arquivo, você pode simular esse cenário endereçando a aplicação como sendo um compartilhamento de rede (por exemplo, `\\MachineName\c$\YourAppPath\YourApp.exe`) e executando-a a partir de seu disco rígido.
  - Se sua aplicação for desenvolvida para ser executada a partir da zona Internet Web, utilize o endereço IP de seu computador (por exemplo, `\\<MachineIpAddress\c$\YourAppPath\YourApp.exe`) para simular esse cenário.

### Utilizar o padrão **Demanda/Assert** para chamadores parcialmente confiáveis

O padrão **Demanda/Assert** é utilizado em módulos totalmente confiáveis para permitir o acesso às operações privilegiadas quando chamadas por chamadores parcialmente confiáveis. Esse padrão é útil quando um chamador parcialmente confiável precisa executar operações privilegiadas de um modo seguro, mas não possui os privilégios necessários. Usando o **Assert**, você garante a confiabilidade dos chamadores de seu código.

---

**Nota:** O padrão **Demanda/Assert** deve ser usado somente quando você compreender por completo os riscos de segurança que ele pode introduzir. Declarar permissões desabilita as verificações normais de permissão do .NET Framework, responsáveis por verificar todos os códigos de chamada da pilha. Desabilitar esse mecanismo pode introduzir uma séria vulnerabilidade de segurança em seu código, e deve ser tentada somente quando você entender perfeitamente suas implicações e já tiver esgotado todas as outras soluções possíveis.

---

Nesse padrão, as chamadas **Demanda** ocorrem antes das chamadas **Assert**. A **Demanda** verifica se o chamador tem a permissão e, a seguir, a **Assert** desabilita a stack walk para a permissão particular de modo que os chamadores não sejam checados para o tempo de execução da linguagem comum para poder verificar se têm as permissões apropriadas.

Para um chamador parcialmente confiável chamar com sucesso um método de módulo totalmente confiável, você pode demandar permissões apropriadas para certificar-se de que o chamador parcialmente confiável não prejudique o sistema e, a seguir, declarar a permissão específica para que possa executar a operação de alto privilégio.

Você deve chamar o **Assert** em seu módulo totalmente confiável antes de fazer a operação privilegiada e chamar o **RevertAssert** em seguida de forma a garantir que o código subsequente em seus

chamados de método não saiam exitosos por engano por que o **Assert** ainda esteja em efeito. Você deve posicionar esse código em uma função particular de modo que o **Assert** seja removido da pilha automaticamente (utilizando um chamado **RevertAssert**) após o retorno do método. É importante tornar esse método particular; assim, um atacante não poderá invocar o método usando um código externo.

### Utilizar o Padrão Demanda/Assert Para Chamadores Parcialmente Confiáveis

Considere o seguinte exemplo.

```
private void PrivilegedOperation()
{
    // Demand for permission.
    new FileIOPermission( PermissionState.Unrestricted ).Demand();
    // Assert to allow caller with insufficient permissions.
    new FileIOPermission( PermissionState.Unrestricted ).Assert();
    // Perform your privileged operation.
}
```

Por padrão, um módulo totalmente confiável não permite chamados a partir de aplicações ou módulos parcialmente confiáveis; tais chamados geram uma exceção de segurança. Para evitar essas exceções, você pode adicionar o **AllowPartiallyTrustedCallersAttribute** (APTCA) ao arquivo montagemInfo.cs gerado pelo Visual Studio .NET conforme descrito abaixo.

```
[montagem: AllowPartiallyTrustedCallersAttribute()]
```

---

**Nota:** Códigos que utilizam APTCA devem ser revisados para garantir que eles não possam ser explorados por nenhum código malicioso parcialmente confiável. Para maiores informações, consulte "APTCA" em "Chapter 8 Code Access Security in Practice" de *Improving Web Application Security: Threats and Countermeasures* em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh08.asp>.

---

### Considerar o Uso de Módulos de Nome Forte

Você pode aumentar a segurança de seus módulos utilizando nomes fortes para eles. Você deve considerar a opção de assinar todos os módulos em sua aplicação com um nome forte, e modificar a política de segurança para poder confiar nesse nome forte. Você pode assinar o módulo com um par de chaves de nome forte utilizando a ferramenta Sn.exe tool. Para mudar a política de segurança manualmente, você pode usar o snap-in do .NET Framework Configuration MMC ou Caspol.exe, uma ferramenta de linha de comando (localizada em %SystemRoot%\Microsoft.NET\Framework\<version>\CasPol.exe).

Seu processo para assinar módulos com chaves particulares deve levar em consideração o seguinte:

- **Use a delayed signing para desenvolvimento.** O processo de construção para a compilação do código por utilizar a delayed signing, utilizando a porção pública do par de chaves de nome forte em vez da chave particular. Para usar a delayed signing, o desenvolvedor pode adicionar os seguintes atributos ao arquivo montagem.cs para seu projeto.

```
[montagem:MontagemKeyFile("publickey.snk")]
```

```
[montagem:MontagemDelaySign(true)]
```

- **Proteja as chaves particulares geradas.** A linha de comando a seguir exhibe o uso da ferramenta de nome forte (Sn.exe), fornecida pelo .NET Framework SDK, para gerar o par de chaves (Keypair.snk) diretamente em um equipamento de armazenamento removível. (No exemplo, a unidade F utilizada é uma unidade USB.)

```
sn -k f:\keypair.snk
sn -p f:\keypair.snk f:\publickey.snk
```

A chave particular (Publickey.snk) é usada para delayed signing pelos desenvolvedores. O par de chaves é armazenado em um local seguro com acesso altamente restrito.

- **Desabilite a verificação para teste.** Para testar um módulo que tenha sido delay signed, você pode registrá-lo em computadores de teste usando Sn.exe. A Tabela 5.1 lista as variações de linha de comando mais comumente utilizadas.

**Tabela 5.1: Variações de Linha de Comando Mais Comumente Utilizadas**

Comando	Descrição
sn -vr montagem.dll	Desabilita a verificação para um módulo específico.
sn -vr *,publickeytoken	Desabilita a verificação para todos os módulos com uma chave pública específica. O asterisco (*) registra todos os módulos delayed signed com uma chave correspondente à chave pública fornecida para pular a verificação.

- **Assinar com a chave particular para lançamento.** Para completar o processo de assinatura, utilize o seguinte comando para assinar com a chave particular.

```
sn -r montagem.dll f:\keypair.snk
```

Os membros da equipe designada devem, então, testar e revisar o módulo antes de liberá-lo para uso da organização.

Para maiores informações sobre delayed signing e o processo explicado nessa seção, consulte os seguintes recursos em Improving Web Application Security: Threats and Countermeasures:

- "Chapter 3 Threat Modeling," em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh03.asp>.
- "Delay Signing" em "Chapter 7 Building Secure Montagens" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh07.asp>.
- "Chapter 5 Architecture and Design Review for Security" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh05.asp>.
- "Chapter 21 Code Review" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh21.asp>.

Consulte o artigo: "Strong Name Signing Using Smart Cards in Enterprise Software Production Environment" em <http://www.dotnetthis.com/Articles/SNandSmartCards.htm>.

### Evitar Conceder Total Confiança para Zonas Restritas

Como uma alternativa rápida para resolver os problemas de segurança com aplicações parcialmente confiáveis, você pode desejar conceder total confiança para zonas restritas, como a zona de Internet ou de Intranet Local. Fazer isso permite que qualquer aplicação seja executada sem verificações de segurança do código de acesso, o que, por sua vez, pode se tornar um problema se a aplicação for de uma fonte maliciosa. Entretanto, se os cenários de implementação forem considerados durante a fase de design, você não deverá abrir a segurança para permitir que aplicações sejam executadas.

## Desenvolver Aplicações Totalmente Confiáveis

Pelo fato de as aplicações parcialmente confiáveis poderem ter pouco acesso aos recursos do sistema, sua aplicação pode vir a requerer mais permissões do que as originalmente atribuídas por padrão de modo a poder operar adequadamente. As aplicações que precisam ser capazes de executar tarefas como abrir aplicações do Office ou o Microsoft Internet Explorer, chamando para o legado componentes COM ou fazendo gravações no sistema de arquivo precisam ser executadas com permissões que habilitem explicitamente essas operações.

Pode ser tentador atribuir sua aplicação como uma aplicação totalmente confiável de forma que ela possa ter a concessão de todas as permissões possíveis. Entretanto, é mais seguro desenvolver e implementar sua aplicação para requerer a quantidade mínima de permissões necessárias para que ela opere apropriadamente. Caso você realmente precise executar sua aplicação como uma aplicação totalmente confiável, siga as seguintes diretrizes:

- Identifique os tipos de recursos que seu módulo precisa acessar e avalie as ameaças potenciais prováveis de ocorrer se o módulo for comprometido.
- Identifique o nível de confiança de seu ambiente-alvo, uma vez que a política de segurança do acesso ao código pode restringir o que seu módulo tem permissão para fazer.
- Reduza a superfície de ataque utilizando o modificador de acesso público somente para classes e membros que formam parte da interface pública do módulo. Sempre que possível, restrinja o acesso para todas as outras classes e membros utilizando o modificador de acesso particular ou protegido.
- Use a palavra-chave **sealed** para evitar a herança de classes que não sejam designadas como uma classe de base, conforme demonstrado no seguinte código.

```
public sealed class NobodyDerivesFromMe
{...}
```

### 112 Guia de Arquitetura e Projeto de Smart Clients

- Onde for possível, use o método de classe declarativo ou atributos de nível de método para restringir o acesso aos membros do grupo Windows especificado, conforme demonstrado no seguinte código.

```
[PrincipalPermission(SecurityAction.Demand, Role=@"DomainName\Windows
Group")]

public sealed class Orders()
{...}
```

- Estabeleça um processo de construção seguro para delayed signing, teste, revisões de segurança e proteção de chaves particulares.

## Sumário

As aplicações Smart Client são aplicações distribuídas. Dessa forma, para gerenciar a segurança de modo efetivo para elas, você precisa considerar a segurança no servidor, no cliente e a conexão de rede entre os dois. As considerações de Smart Client específicas incluem o desenvolvimento de autenticação e autorização segura, validação de dados e proteção de dados sensíveis. Você deve, também, examinar como usar a segurança do acesso ao código de forma a gerenciar a segurança ao nível do código e não ao nível do usuário.

# 6

## Usando Threads Múltiplos

Um thread é uma unidade básica de execução. Um único thread executa uma série de instruções da aplicação, seguindo um caminho único de lógica pela aplicação. Todas as aplicações têm ao menos um thread, mas você pode desenvolver suas aplicações de modo que elas possam usar threads múltiplos, com cada thread executando lógicas separadas. Ao utilizar threads múltiplos em sua aplicação, você pode processar tarefas extensas ou que demandem muito tempo no background. Mesmo em um computador com um único processador, o uso de threads múltiplos pode melhorar significativamente a resposta e aproveitamento de sua aplicação.

Desenvolver sua aplicação de modo a utilizar múltiplos threads pode ser muito complicado, especialmente se você não tomar cuidado com questões de proteção e sincronização. Ao desenvolver sua aplicação Smart Client, você precisa avaliar com atenção onde e como os múltiplos threads devem ser usados para que você possa ganhar máxima vantagem sem criar aplicações que sejam desnecessariamente complexas e difíceis de fazer o debug.

Esse capítulo examina alguns dos conceitos que são mais importantes para o desenvolvimento de aplicações Smart Client com threads múltiplos. Ele descreve alguns dos usos recomendados para threads múltiplos em uma aplicação Smart Client e como implementar essas capacidades

### Threads Múltiplos no .NET Framework

Todas as aplicações .NET Framework são criadas com um único thread, que é utilizado para executar a aplicação. Em aplicações Smart Client, esse thread cria e gerencia a interface do usuário (UI) e é chamado de UI thread.

Utilize o UI thread para todo o processamento, incluindo chamadas de serviço da Web, chamadas de objeto remoto e chamadas em um banco de dados. Entretanto, utilizar o UI thread dessa forma não é geralmente uma boa idéia. Na maioria dos casos, você será incapaz de prever quanto tempo uma chamada para um serviço da Web, objeto remoto ou banco de dados irá levar, e você pode fazer com que o UI trave enquanto o UI thread waits espera por uma resposta.

Criar threads adicionais permite que sua aplicação execute um processamento adicional sem usar o UI thread. Utilize múltiplos threads para evitar o travamento do UI enquanto a aplicação faz uma chamada a um serviço da Web, ou para executar certas tarefas locais em paralelo, de modo a aumentar a eficiência geral de sua aplicação. Na maioria dos casos, você deve considerar a opção de executar quaisquer tarefas não relacionadas ao UI em um thread separado.



## Escolher Entre Chamadas Sincrônicas e Assíncronas

As aplicações podem fazer tanto chamadas síncronas quanto assíncronas. Uma chamada síncrona espera por uma resposta ou valor de retorno antes de proceder. Uma chamada é dita *bloqueada* se não tiver permissão de continuar.

Uma chamada *assíncrona*, ou *não-bloqueada* não espera por uma resposta. Chamadas *assíncronas* são executadas pelo uso de um thread separado. O thread original inicia a chamada assíncrona que, por sua vez, usa outro thread para executar a solicitação enquanto o thread original continua processando.

Com aplicações Smart Client, é importante minimizar chamadas síncronas a partir do UI thread. Ao desenvolver sua aplicação Smart Client, você deve considerar cada chamada que sua aplicação irá fazer e determinar se uma chamada síncrona pode afetar negativamente a resposta e desempenho da aplicação.

Use chamadas síncronas a partir do UI thread somente quando:

- Executar operações que manipulem o UI
- Executar operações pequenas, bem definidas que não apresentem riscos de o UI travar

Use chamadas assíncronas a partir do UI thread quando:

- Executar operações de background que não afetem o UI
- Fizer chamadas para outros sistemas ou recursos localizados na rede
- Executar operações que possam levar um longo tempo para serem concluídas

## Escolher Entre Threads de Foreground ou de Background

Todos os threads no .NET Framework são designados como threads de foreground ou de background. Os dois mantêm apenas uma diferença: threads de background não impedem um processo de terminar. Após todos os threads de foreground pertencentes a um processo tiverem terminando, a CLR (Common Language Runtime) finaliza o processo, terminando quaisquer threads de background que possam ainda estar sendo executados.

Por padrão, todos os threads gerados ao criar e iniciar um novo objeto **Thread** são threads de foreground, e todos os threads que entram em um ambiente de execução gerenciado a partir de um código não-gerenciado são marcados como threads de background. Entretanto, você pode modificar a classificação do thread ao modificar a propriedade **Thread.IsBackground**. Um thread é designado como um thread de background ao definir-se a **Thread.IsBackground** como **true**, e é designado como thread de foreground ao definir-se a **Thread.IsBackground** como **false**.

---

**Observação:** Para maiores informações sobre o objeto Thread, consulte "Usando a Classe de Thread", mais adiante nesse capítulo.

---

Na maioria das aplicações, você optará por definir threads diferentes como sendo de foreground ou de background. Geralmente, você deve definir as threads que escutam passivamente por uma atividade como threads de background, e definir as threads responsáveis pelo envio de dados como threads de foreground, de modo que o thread não seja terminando antes de todos os dados serem enviados.

Você deve utilizar threads de background somente quando tiver certeza de que não haverá efeitos adversos pelo fato do thread ser terminado sem cerimônia pelo sistema. Use um thread de foreground quando o thread estiver executando operações sensíveis ou transacionais que precisem ser concluídas, ou quando você precisar controlar como o thread é finalizado de forma que recursos importantes possam ser liberados.

## Manipulação de Proteção e Sincronização

Às vezes, ao construir aplicações, você cria múltiplos threads que precisam utilizar recursos de chave, como dados ou componentes da aplicação, ao mesmo tempo. Se você não tomar cuidado, um thread pode alterar um recurso, enquanto outro thread está trabalhando com ele. O resultado pode ser que o recurso seja deixado em um estado indeterminado e seja considerado inutilizável. Isso é conhecido como uma *race condition*. Outros efeitos adversos do uso de múltiplos threads sem a consideração necessária do uso compartilhado de recursos incluem deadlocks, estagnação do thread e questões de afinidade do thread.

Para evitar esses efeitos ao acessar um recurso de dois ou mais threads, você precisa coordenar os threads que estão tentando acessar o recurso usando técnicas de proteção e sincronização.

O gerenciamento do acesso de thread a recursos compartilhados utilizando proteção e sincronização é uma tarefa complexa e deve ser evitada sempre que possível, passando os dados entre os threads e não fornecendo acesso compartilhado para uma única instância.

Se você não pode eliminar o compartilhamento de recursos entre os threads, você deve:

- Usar o statement **lock** no Microsoft Visual C#® e o statement **SyncLock** no Microsoft® Visual Basic® .NET para criar uma seção crítica, mas tome cuidado ao fazer chamadas de método do interior de uma seção crítica para evitar deadlocks.
- Use o método **Synchronized** para obter coleções .NET que sejam seguras para thread.
- Use o atributo **ThreadStatic** para criar membros por thread.
- Use uma proteção de dupla verificação ou o método **Interlocked.CompareExchange** para evitar a proteção desnecessária.
- Certifique-se de que o estado estático seja seguro para thread.

Para maiores informações sobre técnicas de proteção e sincronização, consulte "Threading Design Guidelines" em .NET Framework General Reference at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconthreadingdesignguidelines.asp>

## Utilizando Timers

Em algumas situações, você pode não precisar utilizar um thread separado. Se sua aplicação precisar executar operações simples, relacionadas a UI periodicamente, você deve considerar a opção de utilizar um process timer. Process timers são utilizados, às vezes, em aplicações Smart Client para:

- Executar operações em intervalos regulares de tempo.
- Manter velocidades de animação constantes (independentemente da velocidade do processador) ao trabalhar com gráficos.
- Monitorar servidores e outras aplicações para confirmar que eles estejam on-line sendo executados.

O .NET Framework fornece três process timers:

- **System.Window.Forms.Timer**
- **System.Timers.Timer**
- **System.Threading.Timer**

O **System.Window.Forms.Timer** é útil caso queira aumentar os eventos em uma aplicação do Windows Forms. Ele foi otimizado especialmente para trabalhar com o Windows Forms e deve ser utilizado dentro de um Windows Form. Ele é designado a trabalhar em um ambiente single-threaded e

opera sincronizadamente com o UI thread. Isso significa que este timer nunca preempt a execução de um código de aplicação (permitindo que você não chame o **Application.DoEvents**) e é seguro para interagir com o UI.

**System.Timers.Timer** é arquitetado e otimizado para o uso em ambiente de multi-threaded. Diferente do **System.Windows.Forms.Timer**, este timer chama o seu manipulador de evento em um thread trabalhador obtido do thread pool CLR. Você deve assegurar que o manipulador do evento não interaja com o UI nesse caso.

**System.Timers.Timer** expõe uma propriedade **SynchronizingObject** que pode imitar o comportamento do **System.Windows.Forms.Timer**, mas, a menos que você necessite controle mais preciso sobre o timing dos eventos, você deveria utilizar o **System.Windows.Forms.Timer**.

**System.Threading.Timer** é um timer simples de lightweight server-side. Não é, de maneira inerente, thread safe e é mais complicado de usar que outros timers. Este timer é geralmente inadequado em ambientes Windows Forms. Tabela 6.1 Lista as propriedades de cada timer.

**Tabela 6.1: Propriedades de Process Timer**

Propriedades	System.Windows.Forms	System.Timers	System.Threading
Eventos de Threading rodam em que thread?	UI thread	UI or worker thread	Worker thread
Instâncias são thread safe?	Não	Sim	Não
Requer Windows Forms?	Sim	Não	Não
Eventos de timer event podem ser agendados?	Não	Não	Sim

## Quando usar Threads Múltiplos

Multithreading pode ser usado em muitas situações comuns para melhorar significativamente a resposta e usabilidade de sua aplicação.

Você deveria considerar fortemente o uso de threads múltiplos para:

- Comunicar-se através de uma rede, por exemplo, a um Web server, banco de dados ou objeto remoto.
- Executar operações locais de time-consuming que poderia causar o travamento do UI.
- Distinguir tarefas de prioridade variante.
- Melhorar a performance de inicialização da aplicação.

É necessário examinar sua utilização detalhadamente.

### Comunicar-se Através de uma Rede.

Smart-clients podem comunicar-se através de uma rede, de várias maneiras, incluindo:

- Chamadas a objetos remotos, como DCOM, RPC ou .NET remoting
- Comunicados Message-based, como os chamados Web service e solicitações HTTP
- Transações distribuídas

Muitos fatores determinam o quão rápido um serviço de rede responde a solicitação de uma aplicação, incluindo a natureza da solicitação, latência da rede, confiabilidade e frequência de uma conexão, e quão ocupados estão o serviço e o servidor.

Esta imprevisibilidade pode causar problemas de resposta em aplicações single-threaded, e os multithreading são freqüentemente uma boa solução. Você deveria criar um thread separado do UI para todas as comunicações através da rede, e então passe os dados novamente para o UI thread quando a resposta for recebida.

Nem sempre é necessário criar threads separados para comunicação de rede. Se a sua aplicação se comunica através da rede assíncronica, utilizando o Microsoft Windows Message Queuing (Também conhecido como MSMQ) por exemplo, ela não espera por uma resposta antes de seguir. No entanto, mesmo nesse caso, você ainda deveria usar um thread separado para ouvir e processar a resposta quando ela chegue.

## Executar Operações Locais

Mesmo em situações onde o processamento ocorre localmente, algumas operações podem tomar tempo suficiente para afetar negativamente a responsividade de sua aplicação. Tais operações incluem:

- Image rendering
- Manipulação de Dados
- Data sorting
- Busca

Você não deve executar operações como essas no UI thread porque isso pode causar problemas de performance em sua aplicação. Em vez disso, você deveria utilizar um thread adicional para executar essas operações assíncronicamente e prevenir o bloqueio do UI thread.

Em muitos casos, você também deveria arquitetar a aplicação para que ela relate o progresso e o sucesso ou falha das operações em curso no background. Considere permitir ao usuário o cancelamento de operações de background para melhorar a usabilidade.

## Distinguir tarefas de Prioridade Variante

Nem todas as tarefas que a sua aplicação tem que executar terão a mesma prioridade. Algumas tarefas serão críticas, e outras não. Em outros casos, você pode descobrir que um thread é dependente dos resultados de processamento de um outro thread.

Crie threads de diferentes propriedades para refletir as prioridades da tarefa em execução. Por exemplo, você deveria usar um thread de alta prioridade para gerenciar tarefas time-critical, e um thread de baixa prioridade para executar tarefas passivas ou tarefas que não sejam time-sensitive.

## Startup da Aplicação

Sua aplicação frequentemente tem que executar um número de operações em sua primeira execução. Por exemplo, ela pode precisar inicializar seu estado, restaurar ou atualizar dados e abrir conexões para recursos locais. Você deve considerar o uso de um thread separado para inicializar sua aplicação, permitindo que o usuário comece a utilizar a aplicação o quanto antes. Utilizar um thread em separado para a inicialização aumenta a resposta e usabilidade da sua aplicação.

Se você executar a inicialização em um thread separado, você deve prevenir o usuário sobre as operações iniciais que dependem da completa inicialização, atualizando o menu e barra de ferramentas do UI quando a inicialização estiver completa. Você também deve fornecer feedback claro, que notifique o usuário sobre os progressos na inicialização.

## Criando e Utilizando Threads

Existem diversas maneiras de criar e utilizar threads de background no .NET Framework. Você pode usar a classe **ThreadPool** para acessar o pool de threads gerenciados pelo .NET Framework para um dado processo, ou você pode utilizar a classe **Thread** para criar e gerenciar explicitamente um thread. Alternativamente, você pode utilizar objetos delegados ou um proxy Web service para causar processamento específico em um thread não-UI. Essa sessão examina cada um desses diferentes métodos e faz recomendações sobre como cada um deles deve ser usado.

### Utilizando a Classe ThreadPool

Você já deve ter percebido que muitas de suas aplicações se beneficiariam do multithreading. No entanto, O gerenciamento de thread não é somente uma questão de criar um novo thread cada vez que você queira executar uma tarefa diferente. Possuir muitos threads pode fazer com que uma aplicação utilize um número desnecessário de recursos do sistema, particularmente se você tiver um número grande de operações short-running, todas rodando em threads. E também, gerenciar um número grande threads pode ser bastante complexo.

O thread pooling resolve esses problemas fornecendo a sua aplicação um pool de threads trabalhadores que são gerenciados por um sistema, permitindo que você se concentre mais nas tarefas da aplicação do que no gerenciamento dos threads.

Threads podem ser adicionados ao thread pool conforme solicitado por uma aplicação. Quando o CLR começa inicialmente, o thread pool não contém threads adicionais. No entanto, conforme sua aplicação solicita threads, eles são dinamicamente criados e armazenados no pool. Se os threads não são utilizados por algum tempo, eles podem ser eliminados, para que o thread pool encolha e cresça de acordo com as demandas da aplicação.

---

**Nota:** É criado um thread pool por processo, então, se você executar muitos domínios de aplicação dentro de um mesmo processo, um erro em um domínio de aplicação pode afetar todos os outros dentro de um mesmo processo, já que eles utilizam o mesmo thread pool.

---

Um thread pool consiste em dois tipos de threads:

- **Worker threads (Threads trabalhadores).** Os worker threads são parte de um pool de sistema padrão. Eles são threads padrão gerenciados pelo .NET Framework, e a maioria das funções são executadas neles.
- **Completion port threads (Threads de conclusão de porta).** Este tipo de thread é usado para operações I/O assíncronas, utilizando o IOCompletionPorts API.

---

**Nota:** Se a aplicação estiver tentando executar operações I/O com um computador que não tenha a funcionalidade **IOCompletionPorts**, ele reverterá para o uso de worker threads.

---

O thread pool contém um padrão de 25 threads por processador. Se todos os 25 threads estão em uso, requests queue adicionais são disponibilizadas. Cada thread utiliza o padrão de stack size e roda na prioridade padrão.

O exemplo de código abaixo mostra o uso de um thread pool.

```
private void ThreadPoolExample()
{
    waitCallback callback = new waitCallback( ThreadProc );
    ThreadPool.QueueUserWorkItem( callback );
}
```

No código acima, primeiro crie um delegado para o código que você queira executar em um worker thread. O .NET Framework define o delegado **WaitCallback**, que se refere a um método que utiliza um único parâmetro de objeto e retorna sem valores. O método seguinte implementa o código a ser executado.

```
private void ThreadProc( Object stateInfo )
{
    // Do something on worker thread.
}
```

Você solicita um único argumento de objeto para um método **ThreadProc** especificando-o como segundo parâmetro no chamado ao método **QueueUserWorkItem**. No exemplo anterior, nenhum argumento é passado para o método **ThreadProc**, então o parâmetro **stateInfo** será nulo.

Utiliza classe **ThreadPool** quando:

- Tenha um grande número de tarefas pequenas e independentes executadas no background.
- Não necessite ter controle absoluto sobre o thread utilizado para executar uma tarefa.

## Utilizando a Classe Thread

Você pode gerenciar explicitamente as threads utilizando a classe **Thread**. Isso inclui threads criados pelo CLR e aqueles criados fora do CLR que adentram o ambiente gerenciado para executar um código. O CLR monitora todos os threads em seu processo que tenham executado códigos dentro do .NET Framework e utiliza uma instância da classe **Thread** para gerenciá-los.

Sempre que possível, você deve criar threads utilizando a classe **ThreadPool**. No entanto, existem muitas situações onde você precisará criar e gerenciar seu próprios threads em vez de usar a classe **ThreadPool**.

Use um objeto **Thread** quando:

- Necessite que uma tarefa tenha uma prioridade particular.
- Tenha uma tarefa que pode executar por muito tempo (e portanto, bloquear outras tarefas).
- Necessite assegurar que assemblies particulares possam ser acessados por um único thread.
- Necessite ter uma identidade estável associada ao thread.

O objeto **Thread** contém um número de propriedades e métodos que o ajudam a controlar os threads. Você pode definir a prioridade de um thread, questionar o estado atual do thread, abortar threads, bloquear threads temporariamente e executar muitas outras tarefas de gerenciamento.

O exemplo de código a seguir demonstra o uso do objeto **Thread** para criar e começar um thread.

```
static void Main()
{
    System.Threading.Thread workerThread =
        new System.Threading.Thread( SomeDelegate );
    workerThread.Start();
}

public static void SomeDelegate () {Console.WriteLine( "Do some work."
); }
```

Neste exemplo, **SomeDelegate** é um **ThreadStart** delegado — uma referência para o código que será executado no novo thread. O **Thread.Start** submete uma solicitação ao sistema operacional para iniciar o thread.

Se você instanciar um thread dessa maneira, você não poderá passar nenhum argumento para o **ThreadStart** delegado. Se você precisar passar um argumento para um método para ser executado em outro thread, você deveria criar um custom delegate com a assinatura do método solicitado e invocá-lo assincronicamente.

Para mais informações sobre delegados custom, ver “Utilizando Delegados” mais adiante nesse capítulo.

Se você precisar receber atualizações ou resultados de um thread separado, você pode usar o método callback — um delegado que referencia o código a ser chamado depois que thread termina seu trabalho — isso permite que os threads interajam com o UI. Para mais informações, ver “Utilizando Tarefas para Manejar Interações Entre o Thread UI e Outros Threads”, mais adiante, neste capítulo.

## Utilizando Delegados

Um delegado é uma referência (ou um apontador) para um método. Quando você define um delegado, você especifica a assinatura de método que outros métodos devem utilizar caso queiram representar o método. Todos os delegados podem ser invocados tanto sincronicamente como assincronicamente.

Exemplo de código a seguir mostra como declarar um delegado. Este exemplo mostra um cálculo long-running implementado como um método em uma classe.

```
delegate string LongCalculationDelegate( int count );
```

Se o .NET Framework encontra uma declaração delegada como a anterior, ele implicitamente declara uma classe oculta derivada da classe **MultiCastDelegate**, como demonstrado no exemplo de código abaixo.

```
class LongCalculationDelegate : MulticastDelegate
{
    public string Invoke( count );
    public void BeginInvoke( int count, AsyncCallback callback,
        object asyncState );
    public string EndInvoke( IAsyncResult result );
}
```

O tipo **LongCalculationDelegate** é usado para referenciar um método que usa um único parâmetro integer e retorna uma cadeia. O exemplo de código a seguir ilustra um delegado desse tipo que referencia um método específico com a assinatura relevante.

```
LongCalculationDelegate longCalcDelegate =
    new LongCalculationDelegate( calculationMethod );
```

No exemplo, **calculationMethod** é o nome do método que implementa o cálculo que você deseja executar em um thread separado.

Você pode invocar o método referenciado pela instância delegada, tanto sincronicamente como assincronicamente. Para invocar-lo sincronicamente, utilize o seguinte código:

```
string result = longCalcDelegate( 10000 );
```

Esse código utiliza internamente o método **Invoke** definido no tipo delegado acima.

Devido ao fato de o método **Invoke** tratar-se de um chamado síncrono, esse método retorna somente após o retorno do método invocado. O valor de retorno é o resultado do método invocado.

Mais frequentemente, para prevenir o bloqueio do chamado de threads, você escolherá invocar o delegado assincronicamente, usando os métodos **BeginInvoke** e **EndInvoke**. Delegados assincronicos utilizam a capacidade de pooling dos threads do .NET Framework para gerenciamento de threads. O padrão *Chamado Assincronico* (Asynchronous Call) implementado pelo .NET Framework oferece o método **BeginInvoke** para iniciar a operação solicitada em um thread, e fornece o método **EndInvoke** para permitir operações assincronicas sejam completadas e qualquer dado resultante se repassado ao thread chamado. Depois que se completa o processamento de background, você pode invocar um método de callback, dentro do qual você pode chamar um **EndInvoke** para restaurar o resultado da operação assincronica.

Quando você chama o método **BeginInvoke**, ele não espera que se complete o chamado; em vez disso, ele retorna um objeto **IASyncResult** imediatamente, o que pode ser usado para monitorar o progresso de um chamado. Você pode utilizar o **WaitHandle**, membro do objeto **IASyncResult**, para esperar que o chamado assincronico se complete ou usar o número **IsComplete** para conclusão. Se você chamar o método **EndInvoke** antes de o chamado estar completo, ele bloqueará e retornará somente depois de completado chamado. No entanto, você deve ser cuidadoso para não utilizar essas técnicas para esperar pela conclusão de um chamado, porque elas devem bloquear o UI thread. Em geral, o mecanismo de callback é a melhor maneira de ser notificado sobre a conclusão de um chamado.

#### ► Para executar um método referenciado por um delegado assincronicamente

1. Defina um delegado que represente uma operação long-running assincronica, Como mostrado no exemplo a seguir:

```
delegate string LongCalculationDelegate( int count );
```

2. Defina um método que combine com a assinatura delegada. O exemplo de método a seguir simula uma operação time-consuming colocando o thread em modo de espera por milésimos de segundo antes de retornar.

```
private string LongCalculation( int count )
{
    Thread.Sleep( count );
    return count.ToString();
}
```

3. Defina um método de callback que responda ao AsyncCallback delegado definido pelo .NET Framework, como mostrado no exemplo a seguir.

```
private void CallbackMethod( IAsyncResult ar )
{
    // Retrieve the invoking delegate.
    LongCalculationDelegate dlgt =
        (LongCalculationDelegate)ar.AsyncState;
    // Call EndInvoke to retrieve the results.
    string results = dlgt.EndInvoke(ar);
}
```



4. Crie uma instância de um delegado que referencie o método que você deseja chamar assincronicamente e crie um **AsyncCallback** delegado que referencie o método de callback, como mostrado no exemplo de código a seguir.

```
LongCalculationDelegate longCalcDelegate =
    new LongCalculationDelegate( calculationMethod );
AsyncCallback callback = new AsyncCallback( callbackMethod );
```

5. A partir do thread chamado, inicie o chamado assincronico chamando o método **BeginInvoke** no delegado que referencia o código que você deseja executar assincronicamente.

```
longCalcDelegate.BeginInvoke( count, callback, longCalcDelegate );
```

O método **LongCalculation** é chamado em um worker thread. Quando completo, o método **CallbackMethod** é chamado, e os resultados dos cálculos restaurados.

---

**Nota:** O método de callback method é executado em um thread não-UI. Para modificar o UI, você precisa usar técnicas para mudar deste thread para um UI thread. Para mais informações, ver “Utilizando tarefas para Manejar Interações Entre o UI Thread e Outros Threads” , mais adiante, nesse capítulo.

---

Você pode usar delegados padrão para passar parâmetros arbitrários a um método a ser executado em um thread separado (algo que você não pode fazer quando cria threads diretamente, usando o objeto **Thread** ou um thread pool).

Invocar delegados assincronicamente é particularmente útil quando você necessita invocar operações long-running no UI da aplicação. Se os usuários executarem uma operação no UI do qual espera-se que leve muito tempo para ser completada, você não desejará que o UI congele e não possa atualizar-se. Usando um delegado assincronico, você pode retornar controle ao seu principal UI thread para executar essas operações.

Você deve usar um delegado para invocar um método assincronicamente quando:

- Você precisa passar parâmetros arbitrários para um método que você queira executar assincronicamente.
- Você quer utilizar um padrão de *Chamado Assincronico* (Asynchronous Call) fornecido pelo .NET Framework.

---

**Nota:** Para mais detalhes sobre como utilizar **BeginInvoke** e **EndInvoke** para executar chamados assincronicos, ver “Asynchronous Programming Overview” no the .NET Framework Developer's Guide em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrasynchronousprogrammingoverview.asp>.

---

## Chamando Web Services Assincronicamente

Aplicações frequentemente comunicam-se com os recursos de rede utilizando Web services. Em geral, você não deveria chamar um Web service sincronicamente a partir de um UI thread, porque o tempo de resposta de um Web service varia muito, como também todos os tempos de resposta em todas as interações pela rede. Em vez disso, você deve chamar todos os Web services assincronicamente a partir do cliente.

Para ver como chamar Web services assincronicamente, considere o seguinte Web service simples, que permanece inativo por um período de tempo e então retorna um string indicando a conclusão da operação.

```
[WebMethod]
public string ReturnMessageAfterDelay( int delay )
{
    System.Threading.Thread.Sleep(delay);
    return "Message Received";
}
```

Quando você referencia um Web service no sistema de desenvolvimento Microsoft Visual Studio® .NET, ele automaticamente gera um proxy. Um proxy é uma classe que permite invocar seus Web services assincronicamente utilizando o padrão Asynchronous Call implementado pelo .NET Framework. Se você examinar o proxy gerado, você verá os três seguintes métodos.

```
public string ReturnMessageAfterDelay( int delay )
{
    object[] results = this.Invoke( "ReturnMessageAfterDelay",
                                    new object[] {delay});
    return ((string)(results[0]));
}
public System.IAsyncResult BeginReturnMessageAfterDelay( int delay,
                                                         System.AsyncCallback callback, object asyncState )
{
    return this.BeginInvoke( "ReturnMessageAfterDelay",
                             new object[] {delay}, callback, asyncState );
}
public string EndReturnMessageAfterDelay( System.IAsyncResult
                                         asyncResult )
{
    object[] results = this.EndInvoke( asyncResult );
    return ((string)(results[0]));
}
```

O primeiro método é o síncrono, para invocar o Web service. O segundo e o terceiro método tratam-se de métodos assíncronos. Você pode chamar Web service assincronicamente conforme abaixo.

```
private void CallWebService()
{
    localhost.LongRunningService serviceProxy =
        new localhost.LongRunningService();
    AsyncCallback callback = new AsyncCallback( Completed );
    serviceProxy.BeginReturnMessageAfterDelay( callback, serviceProxy,
                                              null );
}
```

Este exemplo é bastante similar ao exemplo de callback assíncronico usando um delegado padrão. Você define um objeto **AsyncCallback** com um método que será invocado quando o Web service retorne. Você invoca um Web service assíncronico com um método que especifique o callback e o próprio proxy, como mostrado no exemplo de código a seguir.

```
void Completed( IAsyncResult ar )
{
    localhost.LongRunningService serviceProxy =
        (localhost.LongRunningService)ar.AsyncState;
    string message = serviceProxy.EndReturnMessageAfterDelay( ar );
}
```

Quando o Web service se completa, o método de callback completo é chamado. Você pode então restaurar seus resultados assíncronicos chamando o **EndReturnMessageAfterDelay** no proxy.

## Utilizando Tarefas para Manipular Interações Entre o UI Thread e Outros Threads

Um dos aspectos mais desafiadores da arquitetura de aplicações multi-threaded é a manipulação das relações entre o UI thread e outros threads. É crítico que os threads de background que você utiliza em sua aplicação não interajam diretamente com a aplicação UI. Se um thread de background tenta modificar um controle no UI de sua aplicação, o controle pode ser deixado em um estado desconhecido. Isso pode causar grandes problemas de difícil diagnóstico em sua aplicação. Por exemplo, um bitmap gerado dinamicamente pode estar impossibilitado de ser utilizado, enquanto outro thread estiver alimentando-o com novos dados. Ou, um componente vinculado a um dataset pode mostrar informação conflitiva enquanto o dataset é atualizado.

Para evitar esses problemas, você nunca deve permitir que outros threads, que não o UI thread, façam modificações nos controles UI, ou em objetos de dados vinculados ao UI. Você deve sempre tentar, e manter, uma separação estrita entre o código UI e os códigos de processamento de background.

Separar o UI thread de outros threads é uma boa prática, mas você ainda precisa passar informações continuamente entre os threads. Sua aplicação multi-threaded precisará tipicamente ser capaz de:

- Obter os resultados de um thread de background e atualizar o UI.
- Relatar o progresso para o UI conforme um thread de background executa seu processamento.
- Controlar os threads de background a partir do UI, por exemplo, deixando que o usuário cancele o processamento de background.

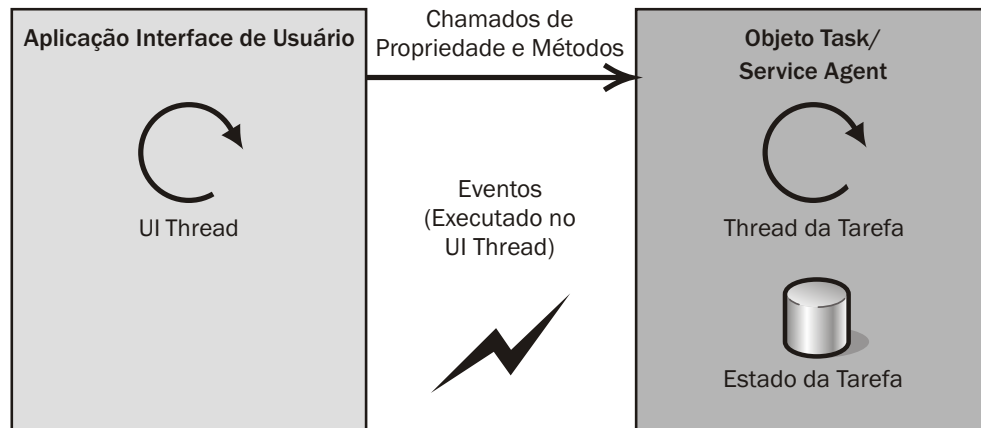
Um modo efetivo de separar o código UI que maneja os threads de background é estruturar sua aplicação em tarefas, e representar cada tarefa usando um objeto que condense todos os detalhes das tarefas.

Uma tarefa é uma unidade de trabalho que usuário espera ser capaz de executar dentro da aplicação. Em um contexto de multithreading, o objeto **Task** condensa todos os detalhes dos threads para que eles estejam claramente separados do UI.

Utilizando o padrão *Task*, você pode simplificar o seu código quando estiver usando threads múltiplos. O padrão *Task*. Claramente separa o código de gerenciamento do thread do código do UI. O UI utiliza propriedades e métodos fornecidos pelo objeto **Task** para executar ações como começar e interromper tarefas, e questionar seu status. O objeto **Task** também pode oferecer um número de eventos, permitindo que a informação de status seja devolvida ao UI.

Todos esses eventos deveriam ser executados no UI thread para que o UI não precise ter conhecimento do thread de background. Você pode simplificar substancialmente as interações dos threads usando o objeto **Task** que é responsável pelo controle e gerenciamento dos threads de background mas executa eventos que podem ser consumidos pelo UI e garantir que estejam no UI thread. Objetos **Task** podem ser reutilizados em várias partes da aplicação, ou até mesmo em outras aplicações.

A Figura 6.1 ilustra a estrutura geral do código quando você utilizar o padrão *Task*.



**Figura 6.1**  
Estrutura do Código utilizando o padrão Task

**Nota:** O padrão Task é usado para executar tarefas de processamentos de background locais em um thread separado ou interagir com um serviço remoto através da rede assincronicamente. Em último caso, o objeto **Task** é frequentemente chamado de agente de serviço. Um agente de serviço pode usar o mesmo padrão como um objeto **Task** e suporta propriedades e eventos que façam sua interação com o UI mais fácil.

Devido ao fato de o objeto **Task** condensar o estado de uma tarefa, use-o para atualizar o UI. Para isso, você pode ter o objeto **Task** executando eventos **PropertyChanged** para o UI thread principal, sempre que ocorrer uma mudança. Esses eventos produzem uma maneira consistente de comunicar as mudanças de valor de propriedade.

Use tarefas para informar o UI thread principal a respeito do progresso ou outras mudanças de estado. Por exemplo, quando uma tarefa torna-se disponível, configure sua flag de permissão, que pode ser usada para permitir os itens de menu correspondentes e os botões da barra de ferramentas. Em sentido inverso, quando uma tarefa torna-se indisponível (por exemplo, por estar em progresso), configure sua flag de permissão para falso, o que faz com que o manipulador do evento no UI thread principal desabilite os itens corretos do menu e os botões da barra de ferramentas.

Você também pode usar tarefas para atualizar objetos de dados vinculados ao UI. Você deve assegurar que quaisquer objetos de dados vinculados aos controles do UI sejam atualizados no UI thread. Por exemplo, se você vincula um objeto **DataSet** ao UI e restaura informações atualizadas de um Web service, utilize os novos dados para seu código UI. O código UI então faz a fusão dos novos dados no **DataSet** vinculado no UI thread.

Você pode usar um objeto **Task** para implementar processamentos de background e controle lógico de threading. Como o objeto **Task** condensa o estado e dados necessários, ele coordena o trabalho necessário para executar a tarefa em um ou mais threads e comunicar mudanças e notificações ao UI da aplicação conforme solicitado. Todo travamento e sincronização necessários podem ser implementados e condensados no objeto **Task**, para que o UI thread não tenha que lidar com esses assuntos.

## Definindo uma Task Class

O exemplo de código a seguir mostra uma definição de classe para uma tarefa que gerencia um cálculo longo.

**Nota:** Apesar de este exemplo ser simples, ele pode ser facilmente estendido para suportar tarefas de background complexas integradas ao UI da aplicação.

```

public class CalculationTask
{
    // Class Members...
    public CalculationTask();
    public void StartCalculation( int count );
    public void StopCalculation();

    private void FireStatusChangedEvent( CalculationStatus status );
    private void FireProgressChangedEvent( int progress );
    private string Calculate( int count );
    private void EndCalculate( IAsyncResult ar );
}

```

A classe **CalculationTask** define um construtor padrão e dois métodos públicos para começar e interromper o cálculo. Ele também define helper methods que ajudam o objeto **Task** a executar eventos no UI. O método **Calculate** implementa o cálculo lógico e roda em um thread de background. O método **EndCalculate** implementa o método de callback, que é chamado após a conclusão do thread de cálculo de background.

Os membros da classe são os seguintes:

```

private CalculationStatus _calcState;
private delegate string CalculationDelegate( int count );
public delegate void CalculationStatusEventHandler(
    object sender, CalculationEventArgs e );
public delegate void CalculationProgressEventHandler(
    object sender, CalculationEventArgs e );
public event CalculationStatusEventHandler CalculationStatusChanged;
public event CalculationProgressEventHandler
CalculationProgressChanged;

```

O membro **CalculationStatus** é uma enumeração que define os três estados em que o cálculo pode estar a qualquer momento.

```

public enum CalculationStatus
{
    NotCalculating,
    Calculating,
    CancelPending
}

```

A classe **Task** oferece dois eventos: um que informa o UI a respeito dos eventos de status do cálculo, e outro para informar o UI dos progressos do cálculo. As assinaturas delegadas são definidas, assim como os próprios eventos. Os dois eventos são executados no *helper methods*. Esses métodos checam o tipo de alvo; se o tipo de alvo for derivado de uma classe **Control**, eles executam os eventos usando o método **Invoke** na classe de controle. Portanto, para consumidor de eventos UI, o evento é garantido para ser chamado no UI thread. O exemplo a seguir mostra o código para execução do evento.

```

private void FireStatusChangedEvent( CalculationStatus status )
{
    if( CalculationStatusChanged != null )
    {
        CalculationEventArgs args = new CalculationEventArgs( status );
        if ( CalculationStatusChanged.Target is
            System.Windows.Forms.Control )
        {
            Control targetForm = CalculationStatusChanged.Target
                as System.Windows.Forms.Control;
            targetForm.Invoke( CalculationStatusChanged,
                new object[] {this, args } );
        }
        else
        {
            CalculationStatusChanged( this, args );
        }
    }
}

```

Este código checa primeiramente se algum consumidor de eventos foi registrado, e caso tenha sido registrado, verifica o tipo de alvo. Se o tipo de alvo é derivado da classe **Control**, o evento é executado utilizando o método **Invoke** para assegurar que seja processado no UI thread. Se o tipo do alvo não for derivado da classe **Control**, o evento é executado normalmente. Eventos são executados da mesma maneira para relatar progresso de cálculo ao UI no método **FireProgressChangedEvent**, como mostrado no exemplo a seguir.

```

private void FireProgressChangedEvent( int progress )
{
    if( CalculationProgressChanged != null )
    {
        CalculationEventArgs args =
            new CalculationEventArgs( progress );
        if ( CalculationStatusChanged.Target is
            System.Windows.Forms.Control )
        {
            Control targetForm = CalculationStatusChanged.Target
                as System.Windows.Forms.Control;
            targetForm.Invoke( CalculationProgressChanged,
                new object[] {this, args } );
        }
        else
        {
            CalculationProgressChanged( this, args );
        }
    }
}

```

A classe **CalculationEventArgs** define os argumentos do evento, para ambos eventos, e contém o status do cálculo e parâmetros de progresso, para que possam ser enviados ao UI. A classe

**CalculationEventArgs** é definida a seguir.

```
public class CalculationEventArgs : EventArgs
{
    public string          Result;
    public int             Progress;
    public CalculationStatus Status;
    public CalculationEventArgs( int progress )
    {
        this.Progress = progress;
        this.Status = CalculationStatus.Calculating;
    }
    public CalculationEventArgs( CalculationStatus status )
    {
        this.Status = status;
    }
}
```

O método **StartCalculation** é responsável pela inicialização do cálculo no thread de background. O **CalculationDelegate** permite que o método **Calculation** seja invocado em um thread de background utilizando o padrão Delegate Asynchronous Call como mostrado no exemplo a seguir.

```
public void StartCalculation( int count )
{
    lock( this )
    {
        if( _calcState == CalculationStatus.NotCalculating )
        {
            // Create a delegate to the calculation method.
            CalculationDelegate calc =
                new CalculationDelegate( Calculation );
            // Start the calculation.
            calc.BeginInvoke( count,
                new AsyncCallback( EndCalculate ), calc );
            // Update the calculation status.
            _calcState = CalculationStatus.Calculating;
            // Fire a status changed event.
            FireStatusChangeEvent( _calcState );
        }
    }
}
```

O método **StopCalculation** é responsável pelo cancelamento do cálculo, como mostrado no seguinte exemplo.

```
public void stopCalculation()
{
    lock( this )
    {
        if( _calcState == CalculationStatus.Calculating )
        {
            // Update the calculation status.
            _calcState = CalculationStatus.CancelPending;
            // Fire a status changed event.
            FireStatusChangedEvent( _calcState );
        }
    }
}
```

Quando o **StopCalculation** é chamado, o estado do cálculo é configurado para **CancelPending**, para sinalizar ao background que pare o cálculo. Um evento é executado para o UI para sinalizar que a solicitação de cancelamento foi recebida.

Ambos os métodos usam a palavra chave **lock** para assegurar que as mudanças no estado variável do cálculo são atômicas, para que sua aplicação não se depare com uma race condition. Os dois métodos executam um evento com status modificado para informar ao UI que o cálculo está iniciando ou interrompendo.

O método de cálculo é definido como segue.

```
private string Calculation( int count )
{
    string result = "";
    for ( int i = 0 ; i < count ; i++ )
    {
        // Long calculation...
        // Check for cancel.
        if ( _calcState == CalculationStatus.CancelPending ) break;
        // Update Progress
        FireProgressChangedEvent( count, i );
    }
    return result;
}
```



---

**Nota:** Por questões de clareza, os detalhes do cálculo foram omitidos.

---

Uma vez que cada passagem é feita através de um loop, o membro de estado de cálculo é verificado para ver se o usuário cancelou o cálculo. Se sim, o loop é abandonado, completando o método de cálculo. Se o cálculo segue, um evento é executando, utilizando o method helper

**FireProgressChanged**, para relatar o progresso ao UI.

Depois de completo o cálculo, o método **EndCalculate** é chamado para terminar o chamado assíncrono através do **EndInvoke**, como mostrado no exemplo a seguir.

```
private void EndCalculate( IAsyncResult ar )
{
    CalculationDelegate del = (CalculationDelegate)ar.AsyncState;
    string result = del.EndInvoke( ar );
    lock( this )
    {
        _calcState = CalculationStatus.NotCalculating;
        FireStatusChangedEvent( _calcState );
    }
}
```

**EndCalculate** reconfigura o estado do cálculo para **NotCalculating**, pronto para que se inicie o próximo cálculo. Ele também executa um evento de estado modificado para que o UI possa ser notificado de que o cálculo foi concluído.

## Utilizando uma Task Class

A classe Task é responsável pelo gerenciamento dos threads de background. Para usar a classe Task, tudo o que você tem a fazer é criar um objeto Task, registrar os eventos que ele execute e implementar a manipulação desses eventos. Como os eventos são executados no UI thread, você não precisa preocupar-se de qualquer assunto de threading em seu código. O exemplo a seguir mostra um objeto Task sendo criado. Neste exemplo, o UI tem dois botões, uma para iniciar o cálculo e outro para interromper, e uma barra de progresso que mostra os progressos atuais de cálculo.

```
// Create new task object to manage the calculation.
_calculationTask = new CalculationTask();

// Subscribe to the calculation status event.
_calculationTask.CalculationStatusChanged += new
CalculationTask.CalculationStatusEventHandler(
    OnCalculationStatusChanged );

// Subscribe to the calculation progress event.
_calculationTask.CalculationProgressChanged += new
CalculationTask.CalculationProgressEventHandler(
    OnCalculationProgressChanged );
```

O manipulador do evento para o status e progressos de cálculo atualiza o UI apropriadamente, por exemplo, atualizando uma barra de controle de status.

```
private void CalculationProgressChanged( object sender,
CalculationEventArgs e )
{
    _progressBar.Value = e.Progress;
}
```

O manipulador do evento **CalculationStatusChanged**, que é mostrado no código a seguir, atualiza o valor de uma barra de progresso para refletir o progresso atual do cálculo. Presume-se que os valores mínimos e máximos da barra de progressos já foram inicializados.

```
private void CalculationStatusChanged( object sender,
CalculationEventArgs e )
{
    switch ( e.Status )
    {
        case CalculationStatus.Calculating:
            button1.Enabled = false;
            button2.Enabled = true;
            break;

        case CalculationStatus.NotCalculating:
            button1.Enabled = true;
            button2.Enabled = false;
            break;

        case CalculationStatus.CancelPending:
            button1.Enabled = false;
            button2.Enabled = false;
            break;
    }
}
```

Nesse exemplo, o manipulador do evento **CalculationStatusChanged** habilita e desabilita os botões de começar e interromper dependendo do status do cálculo. Isso previne que o usuário tente começar um cálculo que já está em progresso e oferece feedback ao usuário sobre o status do cálculo.

O UI implementa manipuladores de evento para cada clique no botão começar e interromper cálculo usando os métodos públicos no objeto **Task**. Por exemplo, um manipulador de evento do botão começar chama o método **StartCalculation** como segue.

```
private void startButton_Click( object sender, System.EventArgs e )
{
    calculationTask.StartCalculation( 1000 );
}
```

Similar, uma botão de parada de cálculo, finaliza o cálculo através da chamada de método **StopCalculation** conforme a seguir.

```
private void stopButton_Click( object sender, System.EventArgs e )
{
    calculationTask.StopCalculation();
}
```

## Sumário

Multithreading é uma parte importante na criação de aplicações Smart Client responsivas. Você deve examinar onde threads múltiplos são apropriados para sua aplicação, buscando conduzir todo processamento que não envolve o UI diretamente em threads separados. Na maioria dos casos, você pode usar a classe **ThreadPool** para criar threads. No entanto, em alguns casos você terá que utilizar a classe **Thread** e em outros você precisará utilizar objetos delegados ou um proxy Web service para causar um processamento específico que ocorra num thread não-UI.

Em aplicações multi-threaded, você deve assegurar que o UI thread seja responsável por todas as tarefas relacionadas ao UI, e que você gerencia a comunicação entre o UI thread e outros threads efetivamente. O padrão *Task* pode ajudar a simplificar essa interação.

# 7

## Implantando e Atualizando Aplicações Smart Client

Aplicações Smart Client executam processamento local nos computadores do cliente, necessitando ser implementadas nesses computadores. No passado, a implementação, atualização, manutenção e instalação de aplicações over time nos computadores do cliente era difícil e problemática. Com o COM, muitos problemas tornaram-no difícil de implementar no computador do cliente, incluindo:

- **Aplicações que foram tightly coupled com o registro.** Instalar uma aplicação COM requeria o registro de classes e tipos de bibliotecas.
- **Aplicações que não tinham autonomia.** Além de registrar classes e tipos no registro, aplicações tipicamente incluíam arquivos compartilhados, localizados no disco, assim como configurações contidas no registro. A aplicação não possuía autonomia; mais, suas partes eram distribuídas em diferentes áreas no computador.
- **Componentes que não podiam ser implementados lado a lado.** Não era possível implementar duas versões diferentes da mesma DLL num mesmo diretório.

Esses problemas eram uma grande barreira para implementação e manutenção efetiva das aplicações do cliente.

O Microsoft® .NET Framework possui um número de características que simplificam o processo de implementação de aplicações .NET Framework. Essas características incluem:

- **Assemblies auto-descritivos.** Os assemblies .NET Framework contêm um metadado que descreve (entre outras coisas) informação de versões, tipos, recursos e detalhes de todos os assemblies referenciados. Isso significa que eles não dependem do registro.
- **Versionamento e suporte lado a lado.** O .NET Framework possui suporte extensivo para versionamento, permitindo que você instale múltiplas versões de uma aplicação e múltiplas versões do .NET Framework, para que eles possam rodar lado a lado.
- **Aplicações isoladas.** Os assemblies .NET Framework podem ser implementados no diretório de aplicação, para uso dessa aplicação específica, e como padrão são mantidos, isolados de outras aplicações. Isso significa que os assemblies não precisam ser aplicados no diretório do Windows ou explicitamente registrados no registro, e reduz a probabilidade que sejam substituídos ou deletados durante a instalação de outras aplicações.

- **Cache de assembly global.** Se você quiser compartilhar códigos entre diferentes aplicações no mesmo computador, você pode implementar componentes ao cache de assembly global. O cache de assembly global permite que diferentes versões de mesmo assembly co-existam. Ao referenciar assemblies no cache de assembly global, você deve especificar o nome qualificado completo do assembly que inclui o key token público e número de versão. Isso ajuda a prevenir o uso não intencional de uma versão ou componente diferente.
- **Vinculação run-time padrão em contra de assemblies build-time para assemblies strong-named.** Pelo padrão, se um assembly é strong named, o .NET Framework vincula a exata versão de seus assemblies dependentes. Isso reduz a fragilidade da aplicação porque o .NET Framework carrega as versões exatas dos assemblies onde foi construído e testado contra. Esse comportamento pode ser explicitamente anulado se solicitado.

Juntas, estas mudanças ajudam a endereçar uma quantidade de características que segmentaram a distribuição e manutenção de aplicações rich clients legadas. Para mais informações sobre como o .NET Framework simplifica a distribuição, em “Simplifying Deployment and Solving DLL Hell with the .NET Framework” No link <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dplywithnet.asp>.

Este capítulo descreve as opções de implementação do próprio .NET Framework, e então examina como implementar aplicações Smart Client baseados no .NET Framework. Há um número de opções para implementação de suas aplicações, e cada uma é discutida, seguido de uma discussão para seleção do método mais apropriado para seu ambiente. Finalmente as opções para implementação de atualizações para a aplicação são examinadas em alguns detalhes.

## Implementando o .NET Framework

A aplicação Smart Client .NET contam com o .NET Framework para funcionar e portanto necessitam que ele seja aplicado no computador do cliente. O .NET Framework é implementado utilizando o .NET Framework Redistributable Package, que pode ser obtido do Microsoft MSDN® ou do Windows Update Web site.

Você também pode obter o redistributable package a partir do CD ou DVD de um produto. O pacote está disponível no .NET Framework SDK, and no DVD Microsoft Visual Studio® .NET 2003.

O .NET Framework Redistributable Package é na verdade um pacote Windows Installer que está envolto em um único arquivo executável auto-removível chamado Dotnetfx.exe. O arquivo executável Dotnetfx.exe inicia o Install.exe, que executa verificações na plataforma, instala a versão 2.0 do Windows Installer se necessário, e então inicia o pacote Windows Installer(.msi file).

Para mais informações sobre o uso do Dotnetfx.exe, ver “.NET Framework Redistributable Package 1.1 Technical Reference” em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetdep/html/dotnetfxref1\\_1.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetdep/html/dotnetfxref1_1.asp).

### Pré-instalação do .NET Framework

Hoje, muitas empresas escolhem implementar o .NET Framework como parte de seu ambiente operacional padrão. Você pode implementar o .NET Framework em sua empresa de duas maneiras:

- **Use tecnologias para incentivar o uso do software no computador do cliente,** como a funcionalidade Group Policy do diretório de serviço Microsoft Active Directory®, ou o Microsoft Systems Management Server (SMS). Usar o software de implementação Group Policy para instalar o pacote em toda rede permite que você assegure que o pacote seja instalado com altos privilégios. Igualmente, usar uma tecnologia de push para empresas, como o SMS permite que você instale o .NET Framework com as permissões necessárias. Para instalar o .NET Framework utilizando o Group

Policy ou o SMS, primeiro você precisa remover o arquivo do Windows Installer do dotnetfx.exe. Para mais detalhes sobre como fazer isso, ver “Redistributing the .NET Framework” em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetdep/html/redistdeploy.asp>.

- **Solicite que usuários finais implementem, eles mesmos, o .NET Framework** usando o Windows Update, ou fazendo o download do .NET Framework de uma rede compartilhada, um Web site interno, ou do Microsoft Web site. Usuários finais precisarão ter privilégios administrativos em seus computadores para implementar o .NET Framework já que o setup do .NET Framework Redistributable Package requer privilégios administrativos para ser instalado.

## Instalando o .NET Framework com uma Aplicação

Nos casos onde você não pode determinar que computadores possuem o .NET Framework pré-instalado, você pode optar por instalar o .NET Framework somente quando necessário — em outras palavras, quando uma aplicação .NET Framework é instalada. Esta abordagem é particularmente útil quando você não sabe as configurações exatas do software dos computadores que serão implementados, e não sabe se o .NET Framework está pré-instalado ou não. Por exemplo, se você é um vendedor de software independente (ISV) desenvolvendo sua aplicação Smart Cliente para venda dentro de uma larga variedade de consumidores, você pode não saber se os seus consumidores têm instalado ou não o .NET Framework.

Para assegurar que o .NET Framework está instalado em sua aplicação, você pode usar a amostra Bootstrapper setup.exe. Essa amostra verifica se o .NET Framework já foi instalado, e se não, a amostra instala o .NET Framework antes de instalar a aplicação.

Para mais informações sobre o uso da amostra Bootstrapper setup.exe, ver Capítulo 3 do Deploying .NET Framework-based Applications em <http://www.microsoft.com/downloads/details.aspx?FamilyId=5B7C6E2D-D03F-4B19-9025-6B87E6AE0DA6&displaylang=en>.

## Implementando Aplicações Smart Client

Durante a arquitetura de suas aplicações Smart Client, você deveria considerar a maneira como essas aplicações serão implementadas. Sempre que possível você deve tentar minimizar o impacto no sistema de qualquer instalação. Esse procedimento permite que você tenha mais controle sobre qualquer mudança na aplicação e resolve problemas de atualização e desinstalação de aplicações. No entanto, algumas vezes você precisará executar instalações mais complexas, por exemplo quando você está reutilizando código de componentes não gerenciados, ou quando você precisa armazenar dados sensíveis de modo seguro no registro.

Um grande número de opções está disponível para sua implementação de aplicações Smart Client. Elas incluem:

- **Implementação No-touch.** Com essa abordagem, você copia os arquivos em um Web server, e o .NET Framework automaticamente fará o download da aplicação e seus assemblies dependentes no cliente quando o usuário clicar em um link.
- **Implementação de Update Stub em Aplicações No-touch.** Com essa abordagem, você utiliza uma implementação no-touch para fazer o download de uma aplicação stub, que então fará o download do restante da aplicação no disco local.
- **Execução de código de um arquivo compartilhado.** Com essa abordagem, Você copia o arquivo para um arquivo compartilhado e roda a aplicação a partir do compartilhamento.
- **Xcopy.** Com essa abordagem, você copia arquivos diretamente no cliente. O .NET Framework permite que a aplicação e todos os seus assemblies dependentes localizem-se em uma única estrutura de diretório, e então você não precisa registrar nada no cliente.

- **Pacotes Windows Installer.** Com essa abordagem, você agrupa os arquivos de sua aplicação em um pacote Windows Installer, e o pacote é então instalado no cliente.

Cada abordagem tem seus pontos fortes e fracos. Para ajudá-lo a determinar a implementação mais apropriada para o seu ambiente, você deve examinar cada um deles mais dethadamente.

## Implementação No-Touch

A implementação no-touch permite que seus usuários acessem sua aplicação em um Web server usando um link URL para a aplicação. Para implementar uma aplicação utilizando implementação no-touch, você simplesmente tem que copiar os arquivos apropriados para um Web server. Quando um usuário navega para o local da aplicação, utilizando um link URL, o Microsoft Internet Explorer faz o download e roda a aplicação. A aplicação e seus assemblies dependentes são baixados para o cliente usando HTTP e são armazenados em um local especial chamado cache de download de assembly. Quando o .NET Framework determina se um assembly no Web server precisa ser baixado ou não, somente o date-time stamp do arquivo é verificado, e não o número de versão do assembly. Se os assemblies no servidor não têm um date-time stamp mais tardio que os no cliente, eles não serão baixados.

Se você utilizar a implementação no-touch para implementar suas aplicações Smart Client, você precisará fornecer ao usuário uma URL que leve ao local da aplicação no Web server. Com essa abordagem, nenhum programa de instalação é necessário no computador do cliente todos os códigos são baixados conforme a necessidade. Sua aplicação é automaticamente atualizada sempre que mudanças ocorrerem no Web server. Se os arquivos mudarem, será feito o download das novas versões quando necessário, como qualquer Web browsing.

A implementação no-touch depende da habilidade do .NET Framework para interagir com o Internet Explorer 5.01 ou mais recente para verificar assemblies .NET que estejam sendo solicitados. Durante uma solicitação, é feito o download do executável para o cache de download. Um processo chamado IEEExec inicia a aplicação em um ambiente isolado e seguro oferecido pela infra-estrutura de código de acesso de segurança do .NET Framework.

---

**Nota:** O cliente tentará rodar a aplicação somente se tiver ambos instalados, o .NET Framework e o Internet Explorer versão 5.01 ou posterior.

---

Se você decidir utilizar a implementação no-touch para implementar uma aplicação que use arquivos de configuração, você pode precisar configurar o diretório Web server directory a fim de permitir o download dos arquivos de configuração da aplicação, já que essa capacidade não está habilitada em seu modo padrão. Assegure-se de habilitar que seja feito o download dos arquivos de configuração somente a partir do diretório onde sua aplicação está localizada; ademais, você pode habilitar arquivos provados de configuração a serem baixados e introduzir um risco de segurança.

---

**Nota:** Arquivos de configuração são baixados duas vezes quando usada a implementação no-touch: a primeira vez para verificar informação sobre vínculos (por exemplo, para controlar a versão exata dos componentes que a aplicação usa) e a segunda para buscar informação de configurações de usuário específicas.

---

Você pode usar a implementação no-touch de dentro de uma aplicação que já tenha sido implementada, baixar e rodar o código usando o método **Assembly.LoadFrom()**. Essa técnica pode ser usada para fazer o download de códigos que mudam frequentemente, ou para fornecer instalação sob demanda de algumas outras funcionalidades.

A implementação no-touch permite que você rode versões localizadas da aplicação. A cultura atual do computador do cliente é a de fazer o download automático dos recursos necessários dos assemblies apropriados para oferecer uma versão localizada da aplicação.

Você pode assegurar aplicações de implementação no-touch usando os mecanismos de segurança oferecidos pelo Web server. Por exemplo, para restringir o acesso à aplicação somente a usuários autorizados em uma intranet, você pode habilitar o Windows Integrated Security no diretório da aplicação no Web server. Para permitir que todos os usuários acessem a aplicação, você pode habilitar acesso anônimo ao diretório da aplicação.

---

**Nota:** Se o seu Web server não permite acessos anônimos ou utilize o Windows Integrated Security para autenticar os clientes, sua aplicação pode não conseguir fazer o download do arquivo de configuração.

---

### Limitações da Implementação No-Touch

A implementação no-touch pode ser útil na implementação de aplicações únicas, ou para implantação de aplicações mais complexas. No entanto, não é uma abordagem apropriada para a completa instalação de aplicações Smart Client mais completas pelas seguintes razões:

- **Configurações de segurança padrão restritas**
- **Funcionalidade offline não confiável**
- **Instalações não transacionadas**

---

**Nota:** A tecnologia ClickOnce na versão 2.0 do .NET Framework removerá a necessidade para fazer manualmente as mudanças na política de segurança para o cliente antes que a aplicação esteja instalada e rodando. A ClickOnce oferecerá um mecanismo configurável para permitir que sejam feitas mudanças na política de segurança automaticamente quando a aplicação for instalada pela primeira vez a partir do Web server. A ClickOnce também fornecerá funcionalidade offline confiável para aplicações Smart Client e permitirá que ela se integrem completamente com o Windows Shell.

---

Esta seção examina as restrições de implementação No-Touchin com maiores detalhes.

### Configurações de Segurança Padrão Restritas

O código de acesso de segurança concede permissões ao aplicativo de acordo com a evidência que o aplicativo apresenta. Pelo padrão, o local da aplicação (a URL por onde foi iniciada) é usada para determinar a permissão que é concedida. A menos que a política de segurança local no computador do cliente tenha mudado, as aplicações de implementação no-touch são parcialmente confiáveis, o que significa que é concedido um número limitado de permissões.

Pelo padrão, uma aplicação Smart Client implementada usando a implementação no-touch não conseguirá executar o que segue:

- Escrever no HD (exceto armazenagem isolada).
- Implementar assemblies no cache de assembly global.
- Implementar ou usar códigos não gerenciados.
- Implementar componentes que solicitem registro ou façam outras mudanças de registro
- Integrar-se com o Windows Shell (especialmente os ícones de instalação no menu Iniciar e o item **Adicionar ou Remover Programas** no Painel de Controle).
- Acessar um Banco de Dados.
- Interagir com qualquer outra aplicação do cliente, como aplicativos do Microsoft Office.
- Acessar Web services ou outros recursos de rede que não estão localizados no mesmo servidor em que a aplicação é implementada.
- Executar outras operações de segurança fora as definidas na zona associada com o local de implementação.

Se a sua aplicação requer mais que um conjunto de permissões padrão e você desejar utilizar a implementação no-touch, você terá que modificar a política de segurança no cliente para conceder à



aplicação permissões que funcionem corretamente. Tal mudança na política de segurança precisa ser propagada para o computador do cliente antes de implementar sua aplicação (por exemplo, usando o Group Policy, um pacote Windows Installer, ou um batch file). Essas solicitações reduzem alguns dos benefícios da abordagem da implementação no-touch. Para mais informações sobre implementação de políticas de segurança, ver “.NET Framework Enterprise Security Policy Administration and Deployment” em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/entsecpoladmin.asp>.

Conforme você arquiteta suas aplicações, é necessário determinar se você pode ir de encontro as especificações de arquitetura de sua aplicação Smart Client e cumprir com as solicitações parciais de confiança para a implementação de uma aplicação no-touch. Em geral, a implementação no-touch e o código em execução de um arquivo compartilhado oferecem soluções de fácil aplicação, mas podem restringir de tal maneira a funcionalidade da aplicação, que ela se torne impraticável para muitas aplicações Smart Client. No entanto, se a sua aplicação não requer nenhuma permissão adicional, a implementação no-touch pode ser o mecanismo de implementação ideal para sua aplicação.

Para mais informações sobre aplicações totalmente confiável e parcialmente confiável, ver Capítulo 4, “Considerações de Segurança.”

### Funcionalidade Offline não Confiável

Um outro problema com a implementação de aplicações Smart Client utilizando a implementação no-touch é que elas não são confiáveis quando offline. Esse problema se deve a diversos fatores:

- **Download retardado dos assemblies.** O download de assemblies é feito sob demanda e armazenado no cache de download de assemblies, que é gerenciado como parte do cache do Internet Explorer. Em alguns casos, quando a aplicação roda on-line, você pode não fazer o download de todas as partes da aplicação, o que afetará a habilidade da aplicação para trabalhar offline.
- **Assemblies podem ser excluídos.** Uma vez que os assemblies localizam-se em uma área gerenciada pelo cache do Internet Explorer, caso o cache seja esvaziado, por qualquer razão, os arquivos de sua aplicação serão excluídos.
- **Aplicações dependentes das configurações offline do Internet Explorer.** Quando tentar rodar uma aplicação offline, você deve configurar o Internet Explorer para que ele rode em modo offline, mesmo que a sua aplicação não seja executada dentro do Internet Explorer. Também, caso você tenha conectividade, mas o Internet Explorer está inadvertidamente configurado para modo offline, nenhuma verificação de atualizações será feita no servidor.

### Instalações Não Transacionadas

Com a implementação no-touch, o download de assemblies é feito quando solicitado a um cache que pode ser esvaziado a qualquer momento. Não é possível então assegurar, em momento algum, que todos os códigos necessários sejam instalados no disco local. Para muitas organizações, essa incerteza é inaceitável para muitas aplicações line-of-business.

### Implementação de Update Stub em Aplicações No-touch

Um dos problemas principais no uso da implementação no-touch é que, pelo padrão, a aplicação roda a partir do cache de download de assemblies e sob confiança parcial, a menos que a política de segurança local seja modificada. Isso pode limitar a funcionalidade de sua aplicação Smart Client, incluindo sua habilidade de funcionar confiavelmente quando offline. Um modo de esquivar esse problema é o uso da implementação no-touch inicialmente para implementar um stub do aplicativo, que, a sua vez, faz o download automático e instala o restante da aplicação no disco local. O stub implementa a aplicação em um local específico no disco, como “C:\Arquivo de programas”, e não está, portanto, sujeito às limitações do cache do Internet Explorer. Quando a aplicação roda, será concedida

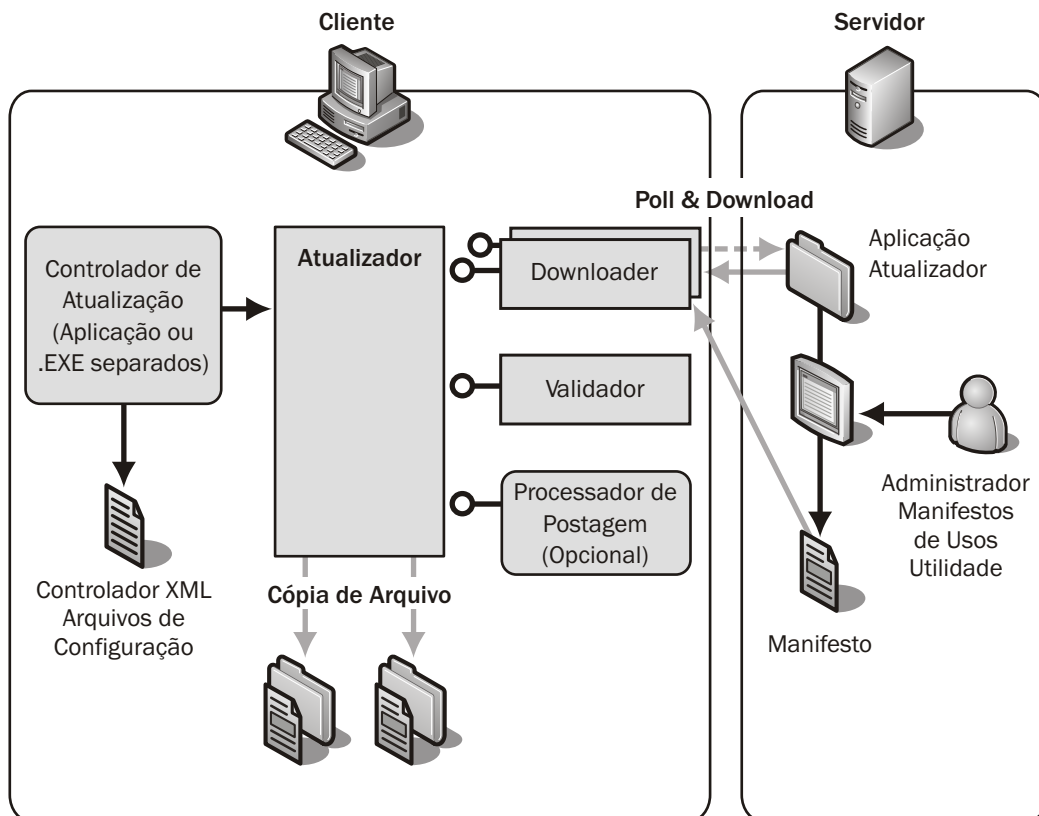
permissão de confiança total porque ela roda a partir do disco local, e pode operar sem as restrições associadas as aplicações de confiança parcial. Um update stub da aplicação pode ser usado também para assegurar que a aplicação seja confiável e atualizada automaticamente caso mudanças no servidor ocorram.

Se você utilizar esse método para implementar sua aplicação, assegure que a política de segurança do .NET Framework do computador do cliente seja modificada para permitir que o stub da aplicação rode com permissões suficiente para fazer o download e armazenar os artefatos no disco local.

Arquitetar update stubs para aplicação pode ser complexo. Para ajudá-lo, a Microsoft criou o Updater Application Block, que você pode utilizar de base para criação de sua própria solução automática de atualização. O Updater Application Block foi criado para:

- Implementar uma solução de atualização pull-based para aplicações .NET Framework.
- Usar técnicas de validação criptográfica para verificar a autenticidade das atualizações de uma aplicação antes de implementá-las.
- Executar tarefas de configuração pós implementação sem a intervenção do usuário.
- Ajudá-lo a criar aplicações que atualizem-se automaticamente, para a última versão disponível.

A arquitetura do Updater Application Block é mostrada na Figura 7.1.



**Figura 7.1**

Arquitetura do Updater Application Block

Para mais informações sobre o Updater Application Block, ver "Updater Application Block for .NET" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp>.

A implementação no-touch com um update stub da aplicação suporta a instalação transacionada de sua aplicação. O Updater Application Block pode ajudar a assegurar que o aplicativo seja instalado com sucesso em sua totalidade. Para executar uma instalação transacionada, você precisará incluir um código que, além de executar atualizações automáticas, verifique se todos os códigos foram instalados no disco local.

Este código pode ser no formato de um manifest file juntamente com o código que determina que cada arquivo do manifest esteja no disco local. Combinar a implementação no-touch com o update stub da aplicação oferece muitos dos benefícios de implementação simplificada, e também se atualiza de acordo com a habilidade de rodar sua aplicação em um ambiente totalmente confiável. Tais benefícios fazem dessa abordagem híbrida uma opção útil na aplicação de muitas aplicações Smart Client. No entanto, não é a escolha ideal em muitas situações. Você ainda precisa conceder ao stub da aplicação permissões suficientes para permitir que ele faça o download do restante da aplicação. Também, aplicações instaladas utilizando essa abordagem não oferecem integração com o Windows Shell (especialmente, integração com o menu **Iniciar** ou o item **Adicionar ou Remover Programas** no Painel de Controle) a menos que você construa essa funcionalidade no stub da aplicação. Finalmente, a implementação e atualização ocorrerão sob o contexto de segurança do usuário. Essa restrição pode causar problemas caso sua aplicação necessite ser escrita no registro, ou em uma parte do sistema de arquivos segurado usuário.

## Executando Códigos de um Arquivo Compartilhado

A execução de um código a partir de um arquivo compartilhado é similar a uma implementação no-touch, exceto pelo fato de que você oferece ao usuário um arquivo compartilhado e não uma URL, de onde aplicar e rodar a aplicação. Códigos que rodam a partir de um arquivo compartilhado são baixados sob demanda e executados no momento apropriado. Como o código roda a partir de uma rede, ele roda como uma aplicação parcialmente confiável, geralmente executando a partir da intranet local e recebendo o conjunto de permissões dessa intranet, a menos que você altere a política de segurança no cliente. Executar códigos a partir de um arquivo compartilhado tem muitas vantagens e desvantagens de uma implementação no-touch, apesar disso, o código não é colocado em cache no cliente como acontece com a implementação no-touch. Devido às restrições de segurança associadas a execução de um código a partir de um arquivo compartilhado, é frequentemente apropriado para a implementação de aplicações Smart Client.

---

**Nota:** Com uma distribuição no-touch, você pode adotar uma abordagem híbrida que combine o código de execução de um arquivo compartilhado com uma automatização atualizada. Para maiores informações, veja “No-Touch Deployment with an Application Update Stub” neste capítulo.

---

## Implementação Xcopy

A implementação **Xcopy** envolve a cópia de todos os arquivos em que consiste a aplicação no computador do cliente de modo que a aplicação possa ser executada. Aplicações Smart Client frequentemente consistem em apenas um ou mais arquivos executáveis, um ou mais DLLs, e um ou mais arquivos de configuração situados em uma hierarquia no diretório. Copiando todos esses arquivos para um outro computador, você instala essencialmente a aplicação. Para desinstalar a aplicação, você apenas remove todos os arquivos do computador.

Nas situações onde você somente necessita modificar o sistema de arquivos para instalar a aplicação, a abordagem **Xcopy** pode ser a melhor opção. Entretanto, como você não terá o controle programático sobre o processo da instalação, a abordagem **Xcopy** não permitirá que você:

- Implemente assemblies no cache de assembly global (e mantenha referências).
- Implemente objetos COM.
- Implemente componentes que necessitem registro ou faça mudanças no registro.
- Integre com o Windows Shell.

Se sua aplicação requerer etapas adicionais de instalação, você pode executar essas etapas manualmente após a cópia dos arquivos. Por exemplo, se você necessitar modificar registro, você pode editar o registro no computador alvo ou importar arquivos \*.reg para assegurar-se de que os ajustes apropriados estejam no lugar. Se você necessitar implementar assemblies no cache de assembly global, você pode usar a utilidade de Gacutil.exe com o switch /ir, que instala os assemblies no cache de assembly global com uma referência seguida. Estas referências podem ser removidas quando o assembly é desinstalado usando o switch /ur.

---

**Nota:** Você também pode usar uma operação drag-and-drop no Windows Explorer para mover os assemblies compartilhados para o folder do cache de assembly global. Entretanto, você deve evitar este método porque ele não implementa a contagem da referência. Sem a contagem de referência, a rotina de desinstalação de uma outra aplicação pode causar a solicitação de um assembly por sua aplicação para ser removida cache de assembly global.

---

A implementação **Xcopy** é apropriado para algumas aplicações Smart Client, mas em muitos casos as etapas adicionais requeridas para que a aplicação funcione corretamente fazem essa abordagem, aparentemente simples, demasiadamente trabalhosa.

## Pacotes Windows Installer

Você pode empacotar sua aplicação para a instalação como um pacote Windows Installer. Esta abordagem lhe oferece a habilidade irrestrita de instalar qualquer coisa no computador alvo, apesar de a aplicação estar limitada no runtime pelo contexto da segurança do usuário final que instala a aplicação.

Os pacotes Windows Installer são muito flexíveis e poderosos, assim que você pode usá-los para instalar aplicações muito complexas que fazem um grande número mudanças de configuração no cliente. Entretanto, eles são apropriados também para aplicações com exigências muito mais simples da instalação. Mesmo que você tenha arquitetado sua aplicação ter o mínimo impacto no cliente quando instalado, você deve considerar o uso de pacotes do Windows Installer, porque eles integram-se com o Windows Shell adicionando ícones no menu **Iniciar** e no desktop, e adicionando a aplicação ao item **Adicionar ou Remover Programas** no Painel de Controle. Esta integração permite que você controle a instalação eficaz e a desinstalação da aplicação quando necessário.

Você pode adicionar qualquer um, ou até mesmo todos, das opções abaixo no pacote Windows Installer:

- Grupos de projeto output
- Arquivos e pastas de serviços
- Assemblies
- Recursos de aplicação
- Fusão de módulos
- Arquivos CAB
- Dependências
- Configurações de registro
- Propriedades de projeto
- Ações Padrão
- Configurações de arquitetura de interface de usuário

Depois de criar um pacote Windows Installer, você tem um grande número de opções para distribuí-lo no computador do cliente, incluindo:

- Usar uma tecnologia de impulso para empresa, como o SMS.
- Usar a funcionalidade Group Policy do Active Directory para publicar ou designar os pacotes.
- Permitir que os usuários instalem o pacote complete a partir de uma mídia, arquivo compartilhado, ou uma URL.

Usar uma tecnologia do impulso para instalar seus pacotes Windows Installer permite que você tenha certo controle centralizado sobre quando e onde a instalação ocorre. Permite também que você controle que grupos dentro da empresa devem ter a aplicação, ou versões particulares da aplicação. Você pode, para o exemplo, assegurar-se de que a instalação ocorra em um determinado momento do dia para um grupo particular de usuários. Entretanto, lembre-se de que você pode precisar de hardware e banda estreita para a rede, dependendo do tamanho de suas aplicações, para assegurar-se de que as distribuições em grande escala trabalhem eficazmente.

Uma das mais significantes vantagens do pacote de Windows Installer é que você pode utilizar uma Diretiva de Grupo (Group Policy) ou SMS, podendo instalar a aplicação sem que o usuário tenha permissões administrativas. Os pacotes de Windows Installer também suportam instalações transacionais. Qualquer uma das mudanças de arquivos e configurações poderão ser instaladas, ou se houver problemas de instalação, o processo poderá dar um roll back, das entradas realizadas pelo Windows Installer.

Uma das vantagens as mais significativas do pacote Windows Installer é que se você usar o Group Policy ou o SMS, você pode instalar a aplicação sem que o usuário solicite permissões administrativas. Os pacotes do Windows Installer também suportam automaticamente instalações transacionadas. Tanto todos os arquivos, como as mudanças de configuração serão instaladas por um pacote do Windows Installer, ou, se houver um problema, a instalação será levada em sua totalidade pelo Windows.

A flexibilidade do pacote do Windows Installers significa que ele pode ser apropriado para instalações de qualquer complexidade, a partir de aplicações que escrevem simplesmente no sistema de arquivos e integram-se com o item **Adicionar ou Remover Programas** no Painel de Controle, aqueles que fazem muitas mudanças de configuração significativas no cliente.

**Nota:** Se você usar pacotes do Windows Installer para implementar sua aplicação; você não tem que usar o mesmo método para implementar atualizações. Em muitos casos é preferível arquitetar sua aplicação de modo a atualizar-se automaticamente depois de instalada. Para mais detalhes sobre a configuração de suas aplicações para atualizações automáticas, veja "Atualizações Automáticas" mais tarde neste capítulo.

## Escolhendo a Abordagem de Implementação Correta

Com tantas escolhas disponíveis para aplicações Smart Client, pode ser um desafio determinar a escolha correta para seu ambiente. Entretanto, as exigências de sua aplicação e as necessidades de seus usuários determinarão normalmente a melhor abordagem.

A tabela seguinte resume as características de cada abordagem de implementação.

**Tabela 7.1: Abordagens de Implementação de Aplicações Smart Client**

	Implementação No touch	Implementação No-touch com update stub da aplicação	Executando Códigos a partir de Arquivo compartilhado	Implementação Xcopy	Pacote Windows Installer
Acesso offline confiável	Não	Sim	Não	Sim	Sim
Full trust funcionalidade da aplicação	Requer Mudanças na política de segurança do cliente	Sim	Requer Mudanças na política de segurança do cliente	Sim	Sim

(continua)

Tabela 7.1: Abordagens de Implementação de Aplicações Smart Client (continuação)

	Implementação No touch	Implementação No-touch com update stub da aplicação	Executando códigos a partir de Arquivo compartilhado	Implementação Xcopy	Pacote Windows Installer
<b>Instalação Usuários Não-Avançados</b>	Sim	Depende dos requerimentos da aplicação	Sim	Sim	Depende dos requerimentos de aplicação e do mecanismo de distribuição da aplicação
<b>Impacto Baixo no Sistema</b>	Sim	Depende dos requerimentos da aplicação	Sim	Sim	Depende dos requerimentos da aplicação
<b>Integração com o Shell do Windows</b>	Não	Não	Não	Não	Sim
<b>Instalação Irrestrita</b>	Não	Não	Não	Não	Sim
<b>Instalação Transacional</b>	Não	Sim	Não	Não	Sim
<b>Necessidade de modificação de diretivas de segurança do .NET Framework do cliente</b>	Sim - Se o cliente precisar executar sob permissões elevadas	Sim - Apenas para o Stub da aplicação	Sim - Se o cliente precisar executar sob permissões elevadas	Não	Não

Em muitos casos a abordagem mais simples é agrupar sua aplicação usando um pacote do Windows Installer. Os pacotes do Windows Installer são altamente flexíveis e permitem que você instale aplicações de qualquer complexidade. Se você usar uma tecnologia de impulso para empresa tais como Group Policy ou SMS para implementar seu pacote Windows Installer, você pode também instalar as aplicações sob um contexto administrativo de segurança, de qualquer maneira do contexto da segurança do usuário. A implementação no-touch com atualização automática também é uma opção viável quando você quer permitir que seus usuários instalem sua aplicação clicando em uma URL, mas você terá que fazer mudanças na política local de segurança do computador alvo para assegurar-se de que sua aplicação stub possa rodar sob full trust.

## Implementando Atualizações Smart Client

Depois de você ter inicialmente implementado suas aplicações Smart Client, seu trabalho não está acabado. As aplicações necessitarão ser atualizadas em horas extras, conforme você atualiza a funcionalidade da aplicação e repara erros ou endereça vulnerabilidades de segurança.

Dependendo da situação, você pode ou não usar a mesma abordagem de implementação para atualizar uma aplicação Smart Client já implementada. Por exemplo, se você inicialmente implementou uma aplicação usando um pacote do Windows Installer, você pode usar atualizações automáticas para implementar atualizações. As especificações do seu ambiente determinarão frequentemente, que metodologia de atualização é a mais apropriada.

Uma exigência comum durante a implementação de atualizações é a habilidade de confederar a infraestrutura da atualização, de modo que as atualizações não funcionem fora de um servidor único ou grupo de servidores controlados por uma única entidade. Por exemplo, se um ISV criar uma aplicação Smart Client que esteja implementada na empresa de um cliente, e o ISV libera uma atualização para a aplicação, a empresa pode desejar fazer o download e testar seu ambiente operacional padrão antes de propagá-lo a todos os computadores que funcionam através da empresa. Confederar a infraestrutura de atualização torna isso possível. Por exemplo, um servidor atualizado poderia existir no site do cliente que é responsável por obter atualizações do ISV. Os clientes que funcionam dentro da empresa obteriam as atualizações do servidor local atualizado, mas somente quando os administradores de TI aprovarem. Esta abordagem também pode ser usada para aumentar o desempenho e a escalabilidade da infra-estrutura de atualizações aliviando a carga de um servidor único ou fazendo de servidor.

Ao implementar atualizações para uma aplicação, você tem as seguintes opções:

- **Implementação No-touch.** Os assemblies atualizados são adicionados ao servidor Web através de download automático feito pelos clientes.
- **Atualizações Automáticas.** A aplicação é configurada para fazer o download automático e instalar as atualizações a partir de um servidor.
- **Atualizações a partir de um arquivo compartilhado.** Os assemblies atualizados são adicionados a um compartilhamento de rede para download automático dos clientes.
- **Atualizações Xcopy.** As atualizações são copiadas diretamente nos clientes.
- **Implementação do Pacote Windows Installer.** O pacote Windows Installer é atualizado, um novo pacote é criado, ou um caminho de pacote é usado para atualizar o cliente.

É útil examinar cada uma das opções mais detalhadamente para que você possa determinar qual é mais apropriada para o seu ambiente.

## Atualizações da Implementação No-touch

Se você usar a implementação no-touch para implementar uma aplicação simples ou partes de uma aplicação mais complexa, você pode atualizar estes conjuntos simplesmente colocando os novos arquivos no Web server. Antes que um assembly seja carregado pela

aplicação, o .NET Framework verifica automaticamente o time stamp do assembly localmente e no servidor Web para ver se o assembly necessita que seja feito um novo download, ou se o assembly pode simplesmente rodar a partir do cache de download de assemblies do usuário.

---

**Nota:** A Implementação no-touch possui um número de limitações que o tornam inadequado para implementação da maioria das aplicações Smart Client. Para mais detalhes, veja "Implementação No-Touch", acima, neste capítulo.

---

Embora a emissão de atualizações usando o método de implementação no-touch seja geralmente muito direto, seus clientes podem ter problemas durante um upgrade devido à falta de suporte para instalações transacionadas. Se você atualizar o diretório enquanto os clientes estiverem usando a aplicação, um cliente pode fazer o download de um código antigo inicialmente e então tentar fazer o download do outro código que tem sido atualizado. Isto pode levar a resultados imprevisíveis e pode fazer com que sua aplicação falhe. A solução mais simples para este problema é implementar qualquer atualização significativa em um diretório separado no servidor Web, e quando a implementação estiver completa, mudar quaisquer ligações à posição nova.

---

**Nota:** Se você escolher implementar sua aplicação usando a implementação no-touch com atualização automática, veja a seguinte seção, "Atualizações Automáticas."

---



## Atualizações Automáticas

Na maioria dos casos, a melhor abordagem para remendar, reacomodar e atualizar aplicações é construir a infra-estrutura de atualização na própria aplicação. Neste caso, a aplicação do cliente pode ser arquitetada para fazer o download e instalar automaticamente atualizações de um servidor, e o administrador de TI libera essas atualizações ao servidor para que os clientes o obtenham. Para conseguir isso, você pode incluir um código a uma aplicação para que ela execute o seguinte:

- Verificação automática por atualizações.
- Fazer o download das atualizações disponíveis.
- Fazer seu próprio upgrade aplicando essas atualizações.

Durante a configuração de sua aplicação para atualizações automáticas, é importante assegurar-se de que todos os arquivos atualizados sejam baixados para o cliente. Isto é particularmente importante quando você atualiza os strong-named assemblies. Assemblies que chamam assemblies strong-named devem especificar a versão do assembly strong-named, então, caso você atualize assemblies strong-named, você deve atualizar também qualquer assembly que os chamem.

Ao configurar as atualizações transacionadas, você pode usar o código para certificar-se de que as atualizações estejam instaladas localmente, verificando-as de encontro a um manifesto.

Frequentemente você decidirá instalar a atualização em um diretório separado, e então, ou remover o diretório original após uma instalação bem sucedida ou deixar o diretório original no lugar para fornecer uma aplicação do recuo.

Para mais detalhes sobre atualizações automáticas e o uso do Updater Application Block, veja "Implementação de Update Stub em Aplicações No-touch" mais acima, neste capítulo.

---

**Nota:** As atualizações automáticas serão simplificadas com o ClickOnce na versão 2.0 do .NET Framework. Como parte de um manifesto de implementação, você poderá especificar se é quando a aplicação deve verificar para ver se há atualizações, juntamente com uma posição alternada de atualização.

---

## Atualização de um Arquivo Compartilhado

Quando você copia os assemblies para um arquivo compartilhado, o download daqueles assemblies no cliente são feitos cada vez que a aplicação roda e não é colocada em cache. Com a implementação no-touch, atualizar uma aplicação que foi originalmente implementada executando um código a partir de um arquivo compartilhado é simplesmente um caso de adicionar o código novo ao arquivo compartilhado. O cliente faz então o download do código novo da próxima vez que ele rode.

## Atualizações Xcopy

Se você distribuir originalmente sua aplicação usando uma técnica de cópia de arquivos, você pode querer implementar atualizações da mesma maneira. Apesar do mecanismo de implementação original, a cópia de arquivos pode ser uma das mais efetivas abordagens para atualizar sua aplicação quando as atualizações forem relativamente simples, tais como modificações para um arquivo de configuração. Em tais casos, implementar uma atualização é simplesmente o caso de copiar os novos arquivos e de remover qualquer arquivo antigo que não seja mais necessário.

Geralmente você pode atualizar os assemblies privados simplesmente copiando a nova versão do assembly sobre a mais antiga. Entretanto, embora você possa usar operações simples de cópia para a implementação inicial de um assembly strong-named, não é possível atualizar automaticamente o assembly strong-named nesta maneira e ter sua aplicação (ou outros assemblies) usando-o automaticamente. O strong name do assembly é armazenado no manifesto de qualquer assembly que o referencia, e versões diferentes de um assembly strong-named são considerados assemblies completamente separados pelo Common Language Runtime (CLR). A menos que você especifique de outra maneira, o CLR carrega a mesma versão do assembly strongnamed em que sua aplicação esteve construída originalmente.



## Atualizações do Windows Installer

O Windows Installer oferece uma solução detalhada para atualizar aplicações .NET Framework. Diversas de suas características são arquitetadas especificamente para resolver problemas de atualização da aplicação.

Os pacotes do Windows Installer têm suporte built-in para o controle de versão. Se você versionar sua aplicação corretamente, o pacote do Windows Installer poder assegurar automaticamente que a atualização aconteça corretamente, e você pode especificar se as versões precedentes da aplicação devem ser removidas quando a aplicação nova é instalada. Se você estiver usando uma tecnologia do impulso, como a SMS, para implementar estas atualizações, você pode também controlar que usuários receberão, e quando, as atualizações. Esta característica é particularmente útil se você estiver testando atualizações com um grupo de pessoas em particular antes de implementar as atualizações mais extensivamente.

Se você planeja fazer um upgrade em sua aplicação usando a tecnologia do Windows Installer, você tem três escolhas para sua execução:

- Construa um patch package (.msp) e aplique-o na aplicação atualmente instalada.
- Faça um upgrade do arquivo existente do Windows Installer.
- Crie um arquivo Windows Installer completamente novo.

No general, se você estiver usando o Windows Installer para implementar atualizações, você deveria usar um patch package ou fazer a atualização do arquivo do Windows Installer já existente. Se você criar um arquivo do Windows Installer inteiramente novo, o sistema operacional Microsoft Windows® não reconhecerá o pacote como uma atualização, e as características de gerenciamento do upgrade do Windows não funcionarão corretamente. Entretanto, em alguns casos, as mudanças são tão extensivas que você pode escolher renunciar a esta funcionalidade e criar um novo arquivo do Windows Installer.

---

**Nota:** Para mais informações sobre a implementação de atualizações usando o Windows Installer, ver Deploying .NET Framework-Based Applications em <http://www.microsoft.com/downloads/details.aspx?FamilyId=5B7C6E2D-D03F-4B19-9025-6B87E6AE0DA6&displaylang=en>.

---

## Escolhendo a Abordagem de Atualização Correta

Em alguns casos, a abordagem de atualização que você escolhe é definida pela abordagem de implementação que você escolheu para sua aplicação. Entretanto, a abordagem mais apropriada é determinada frequentemente pela natureza das atualizações que você está implementando. Para o exemplo, você pode apenas copiar novos arquivos sob os mais antigos, ou você pode querer atualizar a aplicação para que ela rode ao lado da versão antiga. A atualização pode envolver a adição de assemblies novos ao cache de assembly global ou em mudar as informações de configuração no registro. As atualizações se complicam mais se você estiver implementando atualizações nos assemblies strong-named, porque cada assembly que chama o assembly strong-named usará o número da versão na chamada.

A Tabela 7.2 resume as opções disponíveis para a atualização de suas aplicações e as características que cada uma delas suporta.

Tabela 7.2: Abordagens de Atualização para Aplicações Smart Client

	Implementação No touch	Implementação No-touch com update stub da aplicação	Executando códigos a partir de Arquivo compartilhado	Implementação Xcopy	Pacote Windows Installer
<b>Instalação Usuários Não-Avançados</b>	Sim	Depende dos requerimentos da aplicação	Sim	Não	Depende dos requerimentos de aplicação e do mecanismo de distribuição da aplicação
<b>Gerenciamento centralizado de atualizações</b>	Sim	Sim	Sim	Não	Depende do mecanismo de distribuição da aplicação
<b>Atualização baixadas quando a aplicação executa</b>	Sim	Sim	Não	Não	Não
<b>Infra-estrutura de atualização federada</b>	Não	Sim	Não	Não	Sim
<b>Atualizações por usuário / Grupo</b>	Sim	Sim	Não	Não	Depende do mecanismo de distribuição da aplicação
<b>Atualização Transacional</b>	Não	Sim	Não	Não	Sim
<b>Suporte embutido para controle de versão</b>	Não	Não	Não	Não	Sim

Em muitos casos, as atualizações automáticas são a abordagem mais eficaz para implementação de atualizações para sua aplicação. Entretanto, ao implementar atualizações maiores ou atualizações que envolvem mudanças de configuração complexas para o cliente, você pode necessitar usar o Windows Installer, que também tem o benefício do suporte automático de controle de versão.

## Sumário

A implementação de aplicações Smart Client é muito mais fácil do que implementar as aplicações Rich Client no passado, devido às características do .NET Framework. Entretanto, há um grande número de escolhas importantes que você necessita fazer para uma implementação bem sucedida, tanto na maneira como você arquiteta sua aplicação para a uma fácil implementação como em que abordagem de implementação você escolhe para a aplicação e para o próprio .NET Framework.

Na maioria dos casos, a melhor escolha para implementar a aplicação é, ou usado um pacote do Windows Installer, ou usar uma combinação da implementação no-touch e um update stub da aplicação. Você necessitará considerar como manter a aplicação e implementar eficazmente as atualizações depois da implementação. Novamente, na maioria dos casos, a melhor escolha é, provavelmente, Windows Installer, ou atualizações automáticas controladas pela própria aplicação.

# 8

## Performance da Aplicação Smart Client

As aplicações Smart Client podem fornecer uma relação de usuário mais rica e mais responsiva do que as aplicações Web podem, e podem tirar vantagem dos recursos de sistemas locais. Se uma parcela grande da aplicação residir no computador do usuário, a aplicação não necessitará voltas constantes a um servidor Web. Isto pode resultar em um aumento de performance e responsividade. Entretanto, para realizar o potencial total de uma aplicação Smart Client, você deve considerar cuidadosamente questões de performance durante a fase da arquitetura da aplicação. Aderessar questões de performance enquanto você arquiteta e projetar sua aplicação pode ajudá-lo a conter custos prévios e reduzir a probabilidade de ter problemas de performance mais tarde.

---

**Nota:** Melhorar a performance de aplicações Smart Client não limita-se às questões de projeto da aplicação. Há um grande número de passos que você pode tomar durante todo o ciclo de vida da aplicação para fazer com que o código .NET seja bem executado. Embora o runtime da língua comum (CLR) do .NET seja muito eficiente em executar o código, há um grande número das técnicas que você pode usar para aumentar a performance de seu código e impedir que os problemas de performance estejam introduzidos no nível do código. Para mais informação sobre essas questões, veja <http://msdn.microsoft.com/perf>.

---

Definir exigências de desempenho reais e identificar assuntos potenciais na arquitetura de sua aplicação é claramente importante, mas frequentemente os problemas de performance aparecem somente depois que o código foi escrito, e está sendo testado. Neste caso, há ferramentas e técnicas que podem ajudá-lo a localizar problemas de performance.

Este capítulo examina como arquitetar e ajustar suas aplicações Smart Client para uma performance excelente. Discute um grande número de assuntos relacionado ao projeto e arquitetura, incluindo considerações de caching and threading, e examina como realçar a performance de parcelas Windows Forms de sua aplicação. O capítulo também observa algumas das técnicas e das ferramentas que você pode usar para localizar e diagnosticar problemas de desempenho com suas aplicações Smart Client.

### Arquitetura pela Performance

Há muitas coisas que você pode fazer em um projeto da aplicação ou em nível de arquitetura para se assegurar que uma aplicação Smart Client seja bem executada. Você deve assegurar-se de estar criando objetivos reais e mensuráveis para a performance o máximo possível na fase de design, o que permite que você avalie tradeoffs do projeto e forneça o modo de custo mais efetivo para endereçar assuntos de performance. Onde quer que seja possível, os objetivos de performance devem ser

baseados em exigências reais do usuário e do negócio porque estes são influenciados fortemente pelo ambiente em que sua aplicação opera. Performance Modeling é um processo estruturado e que pode ser repetido, e você pode usar para gerenciar e assegurar que sua aplicação vá de encontro aos objetivos do desempenho.

Para mais informações, ver Capítulo 2, “Performance Modeling” em Improving .NET Application Performance and Scalability, em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt02.asp>.

Os Smart Client são geralmente parte de uma aplicação distribuída maior. É importante considerar o desempenho da aplicação Smart Client no contexto da aplicação completa, incluindo todos os recursos localizados na rede que a aplicação do cliente usa. O ajuste fino e a otimização de cada componente em uma aplicação geralmente não é necessário ou possível. Em vez disso, seu ajuste de performance deve ser baseado em prioridades, em tempo, em restrições de orçamento e em riscos. Perseguir uma alta performance para seu próprio benefício geralmente não é uma estratégia de custo efetivo.

Smart Clients também necessitarão coexistir com outras aplicações nos computadores do seu usuário. Conforme você arquiteta suas aplicações Smart Client, você deve levar em consideração o fato que suas aplicações necessitarão compartilhar de recursos de sistema tais como a memória, o tempo de CPU, e a utilização da rede com as outras aplicações no computador do cliente.

---

**Nota:** Informações a respeito da arquitetura de serviços remotos escaláveis e de alto desempenho podem ser encontradas em Improving .NET Performance and Scalability, em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp>. Esse guia contém informação detalhada sobre como otimizar seu código .NET para uma melhor performance.

---

Para projetar Smart Clients com performance eficiente, considere o seguinte:

- **Cache de dados onde apropriado.** O cache de dados pode melhorar dramaticamente a performance da aplicação Smart Client, permitindo que você trabalhe com dados localmente, em vez de ter que recuperá-los constantemente na rede. Entretanto, os dados que são sensíveis ou mudam frequentemente, geralmente não são apropriados para o caching.
- **Otimização de comunicação de rede.** Uma comunicação através de interfaces chatty com os serviços remotos tier com voltas múltiplas para solicitações/respostas para executar uma única operação lógica pode consumir recursos do sistema e da rede, tendo por resultado uma performance pobre da aplicação.
- **Uso eficiente de threads.** Se você usar uma interface de usuário thread (UI) para executar a obstrução de chamadas vinculadas de I/O, o UI pode parecer não responsivo para o usuário. Criar um número grande de threads desnecessários pode resultar em performance deficiente devido ao custo para criar e fechar threads.
- **Uso eficiente de transações.** Se o cliente tiver dados locais, o uso de transações atômicas pode ajudar a assegurar que aqueles dados sejam consistentes. Como os dados são locais, a transação é preferentemente local em vez de distribuída. Para os Smart Clients que estão trabalhando offline, qualquer mudança feita nos dados locais será temporária. O cliente necessita sincronizar as mudanças quando estiver novamente online. Para os dados que não são locais, é possível usar transações distribuídas em alguns casos (por exemplo quando os serviços estão na mesma posição física com boa conectividade e onde o serviço suporta-o). Serviços como Web services e o Message Queuing não suportam transações distribuídas.
- **Otimização do tempo de startup da aplicação.** Startup da aplicação rápidos permitem que o usuário comece a interagir com a aplicação mais rapidamente, o que dá ao usuário uma percepção imediata e favorável do desempenho e da usabilidade da aplicação. Sua aplicação deve ser projetada para que somente os assemblies requeridos sejam carregados no startup da aplicação.

Evita o uso de muitos assemblies, já que carregar cada assembly incorre em um custo de desempenho.

- **Fusão eficiente dos recursos disponíveis.** Decisões pobres de arquitetura, tais como, a implementação de finalizadores quando não é necessário, falha para suprimir a finalização no método Dispose, ou falha na liberação de recursos não gerenciados pode conduzir a um atraso desnecessário para a recuperação de recursos e pode criar os escapes de recurso que degradam o desempenho da aplicação. As aplicações que não liberam corretamente recursos, ou forçam explicitamente a coleção do lixo, podem impedir que o CLR gerencie a memória eficientemente.
- **Otimização de performance do Windows Forms.** As aplicações Smart Client contam com o Windows Forms para fornecer uma relação de usuário rica e responsiva. Há um grande número de técnicas que você pode usar para assegurar-se de que o Windows Forms forneça ótima performance. Essas técnicas incluem a redução da complexidade da interface de usuário, e evitar o carregamento quantidades grandes de dados de uma só vez.

Em muitos casos a performance percebida de sua aplicação, da perspectiva do usuário, é ao menos tão importante quanto a performance real da aplicação. Você pode criar uma aplicação que pareça executar-se muito mais eficientemente ao usuário fazendo determinadas mudanças em sua arquitetura, tal como usar processados assíncrono de background (para manter o UI responsivo), mostrar uma barra de progresso para indicar o progresso das tarefas, e fornecer a opção para que os usuários cancelem tarefas de longa execução. Estas edições são discutidas em mais detalhes em toda esta seção.

## Diretrizes para Colocação de Dados em Cache

O Caching é uma técnica importante para melhorar a performance da aplicação e para fornecer uma relação de usuário responsiva. Você deve considerar as seguintes opções:

- **Colocação de dados restaurados frequentemente em cache para redução de roundtrips.** Se sua aplicação tiver que interagir frequentemente com um serviço de rede para recuperar dados, você deve considerar a colocação de dados em cache no cliente, reduzindo a necessidade de obter repetidamente os dados sobre a rede. Isto pode aumentar a performance substancialmente, fornecendo próximo de acesso instantâneo aos dados, e a remoção do risco de atraso de rede e os outages que podem afetar adversamente a performance de sua aplicação esperta do cliente.
- **Colocação de dados somente para leitura em cache.** Os dados de referência somente para leitura são geralmente um candidato ideal para colocação em cache. Tais dados são usados para fornecer dados para validação e interface de usuário para fins de exposição, tais como as descrições do produto, IDs, e assim por diante. Devido ao fato de que esse tipo dos dados não pode ser mudado pelo cliente, ele pode geralmente ser colocado em cache sem nenhuma manipulação especial no cliente.
- **Colocação em cache de dados a serem enviados a serviços localizados na rede.** Você deve considerar colocar cache os dados que devem ser enviados a um serviço localizado na rede. Por exemplo, se a sua aplicação permitir que os usuários entrem com informação de ordem que consiste em um número de itens de dados recolhidos sobre um número de formulários, considere permitir que o usuário incorpore todos os dados, e então envie em uma chamada da rede no fim do processo de entrada.
- **Minimizar o cache de dados altamente voláteis.** Antes que você possa colocar em cache qualquer dado volátil, você precisa considerar por quanto tempo ele pode permanecer em cache antes que torne-se antigo ou, de outro modo, inútil. Se os dados forem altamente voláteis e sua aplicação contar com informação up-to-date, é provável que os dados possam ser mantidos em cache por um tempo curto, caso possam ser mantidos.

- **Minimizar o cache de dados sensíveis.** Você deve evitar colocar dados sensíveis em cache no cliente porque, na maioria dos casos, você não pode garantir a segurança física do cliente. Entretanto, se você colocar dados sensíveis em cache no cliente, você necessitará codificar os dados, o que têm suas próprias implicações na performance.

Outros assuntos que envolvam a colocação de dados em cache estão cobertos mais detalhadamente no capítulo 2 deste guia. Ver também “Caching” sessão do Improving .NET Application Performance and Scalability, Capítulo 3, “Design Guidelines for Application Performance” (<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt03.asp>) e Improving .NET Application Performance and Scalability, Capítulo 4, “Architecture and Design Review of .NET Application for Performance and Scalability” (<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt04.asp>).

## Diretrizes para Comunicação de Rede

Uma outra decisão que você enfrentará é como arquitetar e trabalhar com serviços de rede, tais como os Web services. Particularmente, você deve considerar a granularidade, a sincronicidade e a frequência da interação com os serviços de rede. Para uma melhor performance e scalability, você deve preferir emitir mais dados em um único chamado, a emitir quantidades menores de dados em diversos chamados. Por exemplo, se sua aplicação permitir que os usuários incorporem múltiplos itens a uma ordem de compra, é melhor coletar dados para todos os itens, e então enviar uma só ordem de compra ao serviço de uma só vez, a emitir os detalhes individualmente em chamadas múltiplas. Adicionalmente à redução do custo geral associado a execução de muitos chamados de rede, isso também reduz a necessidade de gerenciamento complexo do estado dentro do serviço e/ou o cliente.

Sua aplicação Smart Client deveria ser arquitetada para utilizar comunicação assíncrona sempre que possível, já que isso o ajudará a manter a interface de usuário responsiva e executando tarefas em paralelo. Para mais informações sobre como iniciar chamados e restaurar dados assíncronicamente utilizando os métodos **BeginInvoke** e **EndInvoke** ver, “Asynchronous Programming Overview” (<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrasynchronousprogrammingoverview.asp>).

---

**Nota:** Para mais informações sobre arquitetura e construção de aplicações Smart Client ocasionalmente conectadas a uma rede, ver Capítulo 3, “Conectando-se” e Capítulo 4, “Smart Clients Conectados Ocasionalmente.”

---

## Diretrizes de Threading

Usar threads múltiplos dentro de sua aplicação pode ser uma boa maneira de aumentar sua performance e responsividade. Particularmente, você deve considerar o uso de threads para realizar processamentos que podem ser feitos com segurança no background e que não requerem a interação do usuário. Executar tal trabalho de background permite que o usuário continue trabalhando com a aplicação e permite que o thread da interface de usuário da aplicação mantenha sua responsividade.

Bons candidatos para processamento em um thread separado incluem:

- **Inicialização da Aplicação.** Executar uma inicialização longa em um thread de background de modo que o usuário possa interagir o mais cedo possível com sua aplicação, especialmente se uma parte importante ou principal da funcionalidade da aplicação não depende da conclusão dessa inicialização.
- **Chamados de Serviço Remoto.** Fazer todos os chamados remotos sobre a rede em um thread de background separado. É difícil - se não impossível - garantir tempos de resposta para os serviços situados na rede. Executar estes chamados em um thread separado reduz o risco de outages ou de lentidão na rede que afetam adversamente a performance da aplicação.

- **Processamento IO Bound.** Um processamento, como procurar e classificar dados no disco, deve ser feito em um thread separado. Tipicamente, este tipo de trabalho está sujeito as restrições do subsistema do disco I/O, e não à disponibilidade do processador, assim que sua aplicação pode eficazmente manter sua responsividade quando este trabalho for executado no background.

Enquanto os benefícios de performance devido ao uso de threads múltiplos podem ser significativos, é importante notar que threads consomem recursos próprios e usar muitos threads pode criar uma carga no processador, que necessita gerenciar a troca entre os threads. Para impedir isso, considere usar um conjunto de threads em vez de criar e controlar seus próprios threads. Os conjuntos de threads controlarão eficientemente os threads para você, reutilizando threads de objetos existentes e minimizando o custo geral associado com a criação e eliminação do thread.

Se a experiência do usuário for impactada pelo trabalho executado pelos threads de background, você deve manter o usuário sempre informado do progresso do trabalho. Oferecer esse tipo de feedback realça a percepção do usuário sobre o desempenho de sua aplicação e impede que ele suponha que nada está acontecendo. Tente assegurar-se de que o usuário possa cancelar operações longas a qualquer hora.

Você também deve considerar o uso do evento **Idle** do objeto **Application** para execução de operações simples. O evento **Idle** fornece uma alternativa simples para o uso de threads separados para processamento de background. Este evento inicia-se quando a aplicação não tem mais mensagens de interface de usuário para manejar e estão a ponto de

incorporar o estado inativo. Você pode executar operações simples com este evento e tirar da vantagem da inatividade do usuário. Por exemplo:

```
[C#]
public Form1()
{
    InitializeComponent();
    Application.Idle += new EventHandler( OnApplicationIdle );
}

private void OnApplicationIdle( object sender, EventArgs e )
{
}
```

#### [Visual Basic .NET]

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        MyBase.New()

        InitializeComponent()

        AddHandler Application.Idle, AddressOf OnApplicationIdle
    End Sub

    Private Sub OnApplicationIdle(ByVal sender As System.Object, ByVal e As System.EventArgs)

    End Sub
End Class
```

---

**Nota:** Para mais informações sobre o uso de threads múltiplos em Smart Clients, ver Capítulo 6, “Usando Threads Múltiplos.”

---

## Diretrizes de Transações

As transações podem fornecer a sustentação essencial para assegurar-se de que as regras de negócio não sejam violadas e que a consistência dos dados seja mantida. Uma transação assegura que um conjunto de tarefas relacionadas tenha sucesso ou falhe como uma unidade. Você pode usar transações para manter a consistência entre uma base de dados local e outros recursos, including filas do Message Queuing.

Para as aplicações Smart Client que necessitam trabalhar com dados offline em cache quando a conectividade da rede não está disponível, você deve enfileirar os dados transacionais e sincronizá-los com o usuário quando a conectividade da rede estiver disponível.

Você deve evitar o uso de transações distribuídas que envolvam os recursos situados na rede, já que esses cenários podem conduzir a problemas de performance devido a variação de tempo de resposta da rede e dos recursos. Se sua aplicação necessitasse envolver um recurso localizado na rede em uma transação, você deve considerar o uso de transações de compensação, que permitam que sua aplicação cancele um pedido precedente quando uma transação local falha. Embora as transações de compensação possam não ser apropriadas para todas as situações, elas permitem que sua aplicação interaja com os recursos da rede dentro do contexto de uma transação de modo loosely coupled, reduzindo a possibilidade que um recurso sob controle do computador local possa afetar adversamente a performance de sua aplicação.

---

**Nota:** Para mais informações sobre o uso de transações em Smart Clients, ver Capítulo 3, “Conectando-se.”

---

## Otimização do tempo de Startup da Aplicação

Um startup rápido da aplicação permite que o usuário comece a interagir com a aplicação quase imediatamente, dando ao usuário uma percepção imediata e favorável do desempenho e da usabilidade da sua aplicação.

Quando uma aplicação começa, primeiro o CLR é carregado, e a seguir o assembly principal da sua aplicação, seguido por todos os assemblies que são requeridos para resolver os tipos de objetos referenciados no formulário principal da sua aplicação. O CLR não carrega todos os assemblies dependentes neste estágio; carrega somente os assemblies que contêm o tipo de definições para as variáveis do membro em sua classe de formulário principal. Uma vez que estes assemblies são carregados, o compilador just-in-time (JIT) compila o código para os métodos enquanto rodam, começando com o método **Principal**. Outra vez, o compilador JIT não compila todo o código em seu assembly. Em vez disso, o código é compilado como necessário em base de por método.

Para minimizar o tempo de startup de sua aplicação, você deve seguir as seguintes diretrizes:

- **Minimize a variação de membros na classe de formulário principal de sua aplicação.** Isso minimizará o número de tipos que devem ser resolvidos quando o CLR carrega as classes de formulários principais.
- **Minimize o uso imediato de tipos assemblies de classe de base larga, tais como as livrarias XML ou as livrarias do ADO.NET.** Esses assemblies levam tempo para carregar. Usando as classes de configuração da aplicação e as características do trace switch levará a biblioteca de XML. Evite isto se o tempo startup da aplicação for uma prioridade.
- **Carga lenta onde possível.** Busque dados somente quando exigido em vez de carregar upfront e congelar o UI.



- **Arquitete sua aplicação para utilizar menos assemblies.** As aplicações com um grande número de assemblies incorrem no aumento do custo de performance. O custo vem do carregamento de um metadado, acesso a várias páginas da memória em imagens pre-compiladas no CLR para carregar o assembly (se é pré-compilada com a ferramenta Native Image Generator, Ngen.exe), compilação de tempo JIT, verificações da segurança, e assim por diante. Você deve considerar fundir os assemblies baseados em seus padrões de uso para diminuir o custo associado à performance.
- **Evite a arquitetura de classes monolíticas que combinem a funcionalidade de muitos componentes em somente um.** Fatorar a arquitetura em classes menores que necessitam somente ser compiladas quando forem chamadas realmente.
- **Arquitete sua aplicação de modo a fazer chamados paralelos a serviços da rede durante a inicialização.** Chamados aos serviços de rede que podem funcionar paralelamente durante a inicialização, podem tirar vantagem da funcionalidade assíncrona fornecida pelos proxies do serviço. Isto ajuda a liberar o thread em execução e chama serviços simultaneamente para executar as tarefas.
- **Use NGEN.exe para compilar e experimente com assemblies NGen e não-NGen, e determina qual salva o maior número de páginas de conjunto de trabalho.** NGEN.exe, que envia com o .NET Framework, costuma pré-compilar um assembly para criar uma imagem nativa que é então armazenado em uma parte especial do cache de assembly global, pronto para a próxima vez em que seja requerido por uma aplicação. Criar uma imagem nativa de um assembly permite que o assembly carregue e execute mais rapidamente porque o CLR não necessita gerar dinamicamente os códigos e estruturas de dados contidos no assembly. Para mais informações, ver as sessões “Working Set Considerations” e “NGen.exe Explained” no capítulo 5, “Improving Managed Code Performance” of Improving .NET Application Performance and Scalability, at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt05.asp>.

---

**Nota:** Se você usar o NGEN para pré-compilar um assembly, todos os seus assemblies dependentes serão carregados imediatamente.

---

## Manipulação de Recursos Disponíveis

O Common Language Runtime (CLR) usa um coletor de lixo para gerenciar o uso da vida útil e da memória do objeto. Isto significa que os objetos que não estão mais acessíveis são coletados automaticamente pelo coletor de lixo, com a memória que está sendo recuperada automaticamente. Os objetos podem ser não mais acessíveis por um número de razões. Por exemplo, pode não haver nenhuma referência ao objeto ou todas as referências ao objeto podem ser de outros objetos que podem ser coletados como parte do ciclo atual de coleta, quando a coleta automática do lixo livrar seu código da carga associada à exclusão controlando o objeto, significa que seu código não mais terá o controle explícito sobre exatamente quando um objeto é excluído.

Considere as seguintes diretrizes para assegurar-se de que você controle os recursos disponíveis eficazmente:

- **Assegure-se de que o método `Dispose` seja chamado quando o objeto chamado oferece um.** Se o seu código chama objetos que suportam o método **`Dispose`**, você deve assegurar-se de chamar esse método assim que terminar de usar o objeto. Chamar o método **`Dispose`** assegura que recursos não gerenciados sejam liberados pró-ativamente em vez de esperar até que a coleta de lixo ocorra. Alguns objetos oferecem métodos adicionais ao método **`Dispose`** que gerencia recursos, tal como o método **`Close`**. Nesses casos, você deve consultar a documentação de como usar métodos adicionais. Por exemplo, com o objeto **`SqlConnection`**, chamando tanto o **`Close`** ou o **`Dispose`** é suficiente para liberar pró-ativamente a conexão com a base de dados de volta ao pool de conexão. Um modo de assegurar que o **`Dispose`** seja chamado assim que você tiver terminado com o objeto é usar a palavra **`using`** em Visual C# .NET ou os blocos **`Try/Finally`** em Visual Basic .NET.

O fragmento de código abaixo demonstra o uso do **Dispose**.

Exemplo da palavra **using** em C#:

```
using( StreamReader myFile = new StreamReader("C:\\ReadMe.Txt")){  
    string contents = myFile.ReadToEnd();  
    //... use the contents of the file  
} // dispose is called and the StreamReader's resources released
```

Exemplo do bloco **Try/Finally** em Visual Basic .NET:

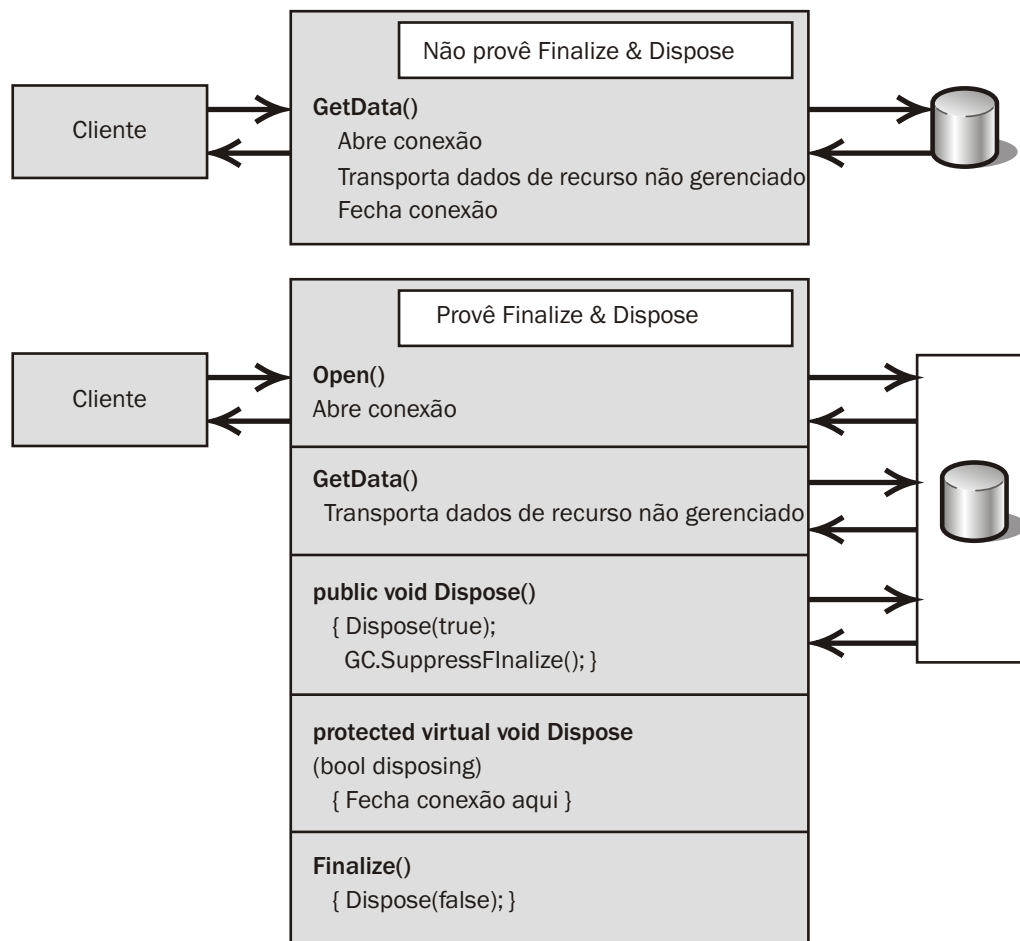
```
Dim myFile As StreamReader  
myFile = New StreamReader("C:\\ReadMe.Txt")  
Try  
    String contents = myFile.ReadToEnd()  
    '... use the contents of the file  
Finally  
    myFile.Close()  
End Try
```

---

**Nota:** Em C# e C++, o método **Finalize** é implementado como destructor. Em Visual Basic .NET, o método **Finalize** é implementado como um finalizador das sub-rotinas do **Finalize** na base de classes do objeto.

---

- **Forneça os métodos **Finalize** e **Dispose** se você possui recursos não gerenciados através dos chamados do cliente.** Se você criar um objeto que acesse recursos não gerenciados em chamados ao método públicos ou protegidos, então a aplicação necessita controlar a vida útil dos recursos não gerenciados. Na figura 8.1, o primeiro caso é de um chamado a recursos não gerenciados onde o recurso é aberto, transferido e fechado. Neste caso, seu objeto não necessita fornecer os métodos **Finalize** e **Dispose**. No segundo caso, o recurso não gerenciado é preso através das chamadas do método; conseqüentemente, seu objeto deve fornecer métodos **Finalize** e **Dispose** para que o cliente possa explicitamente liberar o recurso tão logo o cliente tenha terminado de usar o objeto.



\* Visão simplificada de um Dispose Pattern

**Figura 8.1**

*Uso dos métodos de chamado Dispose e Finalize*

A coleta de lixo é geralmente boa para a performance total porque favorece a velocidade sobre o uso da memória. Os objetos necessitam ser suprimidos somente quando os recursos de memória estão baixos; se não, todos os recursos disponíveis da aplicação são usados para o benefício de sua aplicação. Entretanto, se seu objeto mantém uma referência a recursos não gerenciados, tais como window handle, arquivo, objetos GDI e as conexões de rede, uma performance melhor pode ser conseguida se o programador liberar explicitamente estes recursos quando não estão mais sendo utilizados. Se você mantém recursos não gerenciados através dos chamados do método do cliente, então o objeto deve permitir que o autor do chamado controle explicitamente recursos usando a interface **IDisposable**, que fornece o método **Dispose**. Ao implementado o **IDisposable**, o objeto informa que, preferentemente, se pode pedir para limpar, em vez de esperar pela coleta de lixo. O autor do chamado de um objeto que implemente o **IDisposable** chama simplesmente o método **Dispose** quando terminado com o objeto, para que assim possa liberar o recurso como apropriado.

Para mais informações sobre como implementar o **IDisposable** em um de seus objetos, ver capítulo 5, "Improving Managed Code Performance," em *Improving .NET Application Performance and Scalability*, em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt05.asp>.

---

**Nota:** Se seu objeto descartável deriva-se de um outro objeto que também implemente a interface **IDisposable**, você deve chamar o método **Dispose** da classe para permitir a limpeza em seus recursos. Você deve chamar também o **Dispose** em todos os objetos que são possuídos por seu objeto que implementa a interface **IDisposable**.

---

O método **Finalize** também permite que seu objeto libere explicitamente todos os recursos que têm uma referência de quando o objeto está sendo excluído. Devido à natureza não-determinista do coletor de lixo, em alguns casos o método **Finalize** pode não ser chamado por muito tempo. De fato, pode nunca ser chamado se sua aplicação terminar antes que o objeto seja excluído pelo coletor de lixo. Entretanto, é importante usar o método **Finalize** como uma estratégia de backup caso o autor do chamado não chame o método **Dispose** explicitamente (tanto o método **Dispose**, como o **Finalize** compartilham do mesmo código de limpeza do recurso). Deste modo, o recurso provavelmente liberado em algum ponto, mesmo que isso ocorra mais tarde que considerado ótimo.

---

**Nota:** Para assegurar-se de que o código de limpeza dentro dos métodos **Dispose** e **Finalize** não seja chamado duas vezes, você deve chamar **GC.SuppressFinalize**, que diz ao coletor de lixo para não chamar o método **Finalize**.

---

O coletor de lixo implementa o método **Collect**, que força apagar os arquivos que estão pendentes para exclusão. Este método não pode ser chamado da sua aplicação, pois executa como uma thread de alta prioridade. O ciclo pode parar todas as Threads de Interface do Usuário (UI), resultando uma tela sem retornos.

Para mais informações, ver “Garbage Collection Guidelines,” “Finalize and Dispose Guidelines,” “Dispose Pattern,” e “Finalize and Dispose Guidelines” em Improving .NET Application Performance and Scalability em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt05.asp>.

## Otimização da Performance do Windows Forms

Windows Forms fornece uma interface de usuário rica para sua aplicação Smart Client e há um número de técnicas que você pode usar para ajudar a assegurar que o Windows Forms forneça ótima performance. Antes de discutir técnicas específicas, é útil rever algumas diretrizes de alto nível que podem aumentar a performance do Windows Forms substancialmente.

- **Seja cauteloso com as criações handle.** Windows Forms virtualiza as criações handle (ou seja, cria e recria objetos de handle da janela dinamicamente). Criar objetos handle pode ser caro; portanto, evite fazer mudanças de border style desnecessárias ou mudanças de MDI.
- **Evite criar aplicações com controles child.** O sistema operacional Microsoft® Windows® tem um limite de 10.000 controles por processo, mas você deve evitar ter muitas centenas dos controles em um formulário enquanto cada controle consome recursos da memória.

O restante desta seção discute técnicas mais específicas que você pode usar para otimizar a performance da interface de usuário da sua aplicação. O restante desta seção discute técnicas mais específicas que você pode usar para otimizar a performance da interface de usuário da sua aplicação.

### Usando o BeginUpdate e o EndUpdate

Um grande número de controladores do Windows Forms (por exemplo os controles do **ListView** e do **TreeView**) implementam os métodos **BeginUpdate** e **EndUpdate**, que evitam uma nova pintura de controles quando as propriedades subjacentes dos dados ou do controle forem manipuladas. Usar os métodos **BeginUpdate** e **EndUpdate** permite que você faça mudanças significativas em seus controles e evite que os controles façam novas pinturas constantemente enquanto as mudanças são aplicadas. Tais novas pinturas conduzem a uma degradação significativa de performance e uma interface de usuário hesitante e não responsiva. Para o exemplo, se sua aplicação tiver um controle da árvore que requeira um número grande de artigos de nó a serem adicionados, você deve chamar o **BeginUpdate**, adicionar todos os itens necessários, e chamar então o **EndUpdate**. O exemplo de código seguinte mostra uma árvore de controle que está sendo usada para indicar uma representação hierárquica de um número de clientes com sua informação de ordem.

**[C#]**

```
// Suppress repainting the TreeView until all the objects have been
created.
treeView1.BeginUpdate();

// Clear the TreeView.
treeView1.Nodes.Clear();

// Add a root TreeNode for each Customer object in the ArrayList.
foreach( Customer customer2 in customerArray
{
    treeView1.Nodes.Add( new TreeNode( customer2.CustomerName ) );
    // Add a child TreeNode for each Order object in the current
    Customer.
    foreach( Order order1 in customer2.CustomerOrders )
    {
        treeView1.Nodes[ customerArray.IndexOf(customer2) ].Nodes.Add(
            new TreeNode( customer2.CustomerName + "." + order1.OrderID
                ) );
    }
}

// Begin repainting the TreeView.
treeView1.EndUpdate();
```

**[Visual Basic .NET]**

```
' Suppress repainting the TreeView until all the objects have
been created.
TreeView1.BeginUpdate()

' Clear the TreeView
TreeView1.Nodes.Clear()

' Add a root TreeNode for each Customer object in the ArrayList
For Each customer2 As Customer In customerArray
    TreeView1.Nodes.Add(New TreeNode(customer2.CustomerName))

    ' Add a child TreeNode for each Order object in the current
    Customer.
    For Each order1 As Order In customer2.CustomerOrders
        TreeView1.Nodes(Array.IndexOf(customerArray,
            customer2)).Nodes.Add( _
            New TreeNode(customer2.CustomerName & "." &
                order1.OrderID))
    Next
Next

Next

' Begin repainting the TreeView.
TreeView1.EndUpdate()
```

Você deve usar os métodos **BeginUpdate** e **EndUpdate** mesmo quando você não espera que muitos objetos sejam adicionado ao controle. Na maioria de casos, você não estará ciente do número exato de artigos a serem adicionados até o runtime. Conseqüentemente, para lidar elegantemente com uma quantidade extraordinariamente grande de dados e para as exigências futuras, você deve chamar sempre os métodos **BeginUpdate** e **EndUpdate**.

---

**Nota:** Chamar o método **AddRange** de muitas das classes da coleção usadas pelos controles do Windows Forms chamará automaticamente o **BeginUpdate** e o **EndUpdate**..

---

### Usando o **SuspendLayout** e o **ResumeLayout**

Um grande número de controles do Windows (por exemplo, os controles do **ListView** e do **TreeView**) implementam os métodos **SuspendLayout** e **ResumeLayout**, que impedem que o controle crie eventos múltiplos de layout quando os controles child estiverem sendo adicionados.

Se seus controles programaticamente adicionarem e removerem controles child ou executarem layouts dinâmicos, então você deve chamar os métodos **SuspendLayout** e **ResumeLayout**. O método **SuspendLayout** permite que as ações múltiplas sejam executadas em um controle sem executar uma disposição para cada mudança. Por exemplo, se você redimensiona e move um controle, cada operação criaria um layout separado. Estes métodos operam de maneira similar aos métodos **BeginUpdate** e **EndUpdate** e fornecem os mesmos benefícios em termos de estabilidade da interface de usuário e performance.

O exemplo abaixo programaticamente adiciona botões à parent form:

```
[C#]
private void AddButtons()
{
    // Suspend the form layout and add two buttons.
    this.SuspendLayout();
    Button buttonOK = new Button();
    buttonOK.Location = new Point(10, 10);
    buttonOK.Size = new Size(75, 25);
    buttonOK.Text = "OK";

    Button buttonCancel = new Button();
    buttonCancel.Location = new Point(90, 10);
    buttonCancel.Size = new Size(75, 25);
    buttonCancel.Text = "Cancel";

    this.Controls.AddRange(new Control[] {buttonOK, buttonCancel});
    this.ResumeLayout();
}
```

### [Visual Basic .NET]

```
Private Sub AddButtons()
    ' Suspend the form layout and add two buttons
    Me.SuspendLayout()
    Dim buttonOK As New Button
    buttonOK.Location = New Point(10, 10)
    buttonOK.Size = New Size(75, 25)
    buttonOK.Text = "OK"

    Dim buttonCancel As New Button
    buttonCancel.Location = New Point(90, 10)
    buttonCancel.Size = New Size(75, 25)
    buttonCancel.Text = "Cancel"

    Me.Controls.AddRange(New Control() {buttonOK, buttonCancel })
    Me.ResumeLayout()
End Sub
```

Você deve usar os métodos **SuspendLayout** e **ResumeLayout** sempre que você adicionar ou remover controles, executar um layout dinâmico dos controles child, ou ajustar quaisquer propriedades que afetem a disposição do controle, tal como o tamanho, posição, link ou as propriedades de doca.

### Manipulando Imagens

Se sua aplicação mostra um grande número de arquivos de imagem, tais como o .jpg e os arquivos .gif, então você pode melhorar a performance do vídeo significativamente pre-renderizando as imagens em formato bitmap.

Para usar esta técnica, carregue primeiramente a imagem do arquivo e então converta esta imagem para bitmap usando o formato PARGB. A seguinte amostra de código carrega um arquivo a partir do disco e usa então a classe para converter a imagem em um formato pré-multiplicado, RGB alfa-blended.

Por exemplo:

**[C#]**

```
if ( image != null && image is Bitmap )
{
    Bitmap bm = (Bitmap)image;
    Bitmap newImage = new Bitmap( bm.Width, bm.Height,
        System.Drawing.Imaging.PixelFormat.Format32bppArgb );
    using ( Graphics g = Graphics.FromImage( newImage ) )
    {
        g.DrawImage( bm, new Rectangle( 0,0, bm.Width, bm.Height ) );
    }
    image = newImage;
}
```

**[Visual Basic .NET]**

```
If Not(image Is Nothing) AndAlso (TypeOf image Is Bitmap) Then
    Dim bm As Bitmap = CType(image, Bitmap)
    Dim newImage As New Bitmap(bm.Width, bm.Height, _
        System.Drawing.Imaging.PixelFormat.Format32bppArgb)

    Using g As Graphics = Graphics.FromImage(newImage)
        g.DrawImage(bm, New Rectangle(0, 0, bm.Width, bm.Height))
    End Using
    image = newImage
End If
```

### Use o Paging e o Lazy Loading

Na maioria de casos, você deve recuperar ou exibir dados somente quando necessário. Se sua aplicação precisa recuperar e exibir muita informação, você deve considerar quebrar os dados em páginas e exibir os dados em uma página de cada vez. Isto permite que sua interface de usuário funcione melhor porque não tem que exibir uma quantidade grande de dados. Além, isto pode melhorar a usabilidade de sua aplicação porque o usuário não é confrontado com uma abundância de dados de uma só vez e pode navegar mais facilmente para encontrar os dados exatos que ele ou ela necessita.

Para o exemplo, se sua aplicação exibe dados do produto de um catálogo de produtos grande, você pode exibir os artigos em ordem alfabética com todos os produtos que começam com o "A" exibidos em

uma página e todos os produtos que começam com a "B" na página seguinte. Você poderia então permitir que o usuário navegue diretamente até a página apropriada para ele ou ela não precise rolar toda a página para alcançar os dados que ele ou ela necessita.

Paginar os dados dessa maneira também pode permitir que você busque os dados no background enquanto necessário. Por exemplo, você pode precisar buscar somente a primeira página da informação para exibição e permitir que o usuário interaja com ela. Você pode então deixar a página de dados seguinte pronta no background para quando o usuário a necessite. Esta técnica pode ser particularmente eficaz quando combinada com dados em cache.

Você também pode aumentar a performance de sua aplicação Smart Client usando técnicas de Lazy Load. Em vez de carregar os dados ou recursos que você pode necessitar em algum momento futuro imediatamente, você carrega-os somente quando necessário. Você pode usar o Lazy Load para aumentar a performance de sua interface de usuário ao construir grandes listas ou estruturas de árvores. Neste caso, você pode carregar os dados quando o usuário necessitar, para o exemplo quando uma árvore é expandida.

### Otimizar a Velocidade de Exibição

Você pode otimizar a velocidade de exibição da sua aplicação de muitas maneiras diferentes, de acordo com as técnicas que você estiver usando para exibir os controles da interface de usuário e os formulários de aplicação.

Quando sua aplicação começa, você deveria considerar sua exibição de modo simples como uma interface de usuário. Isto diminuirá o tempo de startup e apresentará uma interface de usuário organizada e fácil de usar para o usuário. Também, você deve tentar evitar referenciar classes e carregar todos os dados que não sejam imediatamente necessários no startup. Isto melhorará o tempo da iniciação da aplicação e do .NET Framework e melhorará a velocidade de exibição da aplicação.

Quando você necessita exibir uma caixa de diálogo ou formulário, você deve mantê-los escondidos até que estejam prontos para exibição, para reduzir a quantidade de pintura necessária. Isto ajudará a assegurar que o formulário somente seja exibido quando estiver sendo inicializado.

Se sua aplicação tiver controles que contenham controles child cobrindo toda a área de superfície do cliente, você deve considerar ajustar o estilo do controle de background para opaco. Isso evita que você tenha que redesenhar o background do controle em cada evento da pintura. Você pode ajustar o estilo do controle usando o método **SetStyle**. Use a enumeração de **ControlsStyles.Opaque** para especificar um estilo opaco no controle.

Você deve evitar re-pintar desnecessariamente os controles. Uma abordagem é esconder os controles enquanto você ajusta suas propriedades. As aplicações que têm o códigos de desenho complexos no evento **OnPaint** podem re-desenhar apenas a região inválida do formulário, em vez de pintar o formulário inteiro. O parâmetro **PaintEventArgs** do evento **OnPaint** contém uma estrutura **ClipRect** que indica que parte da janela é inválida. Isto reduz o tempo que o usuário espera para ver uma exibição completa.

Use otimização padrão de desenho, tal como o clipping, o double buffering ou o **ClipRectangle**. Isto ajudará também a melhorar a performance de exibição de sua aplicação Smart Client impedindo operações de desenho desnecessárias para partes da exibição que não são visíveis ou que requerem novo desenho. Para mais informação sobre como realçar a performance de pintura, veja *Painting techniques using Windows Forms for the Microsoft .NET Framework* em <http://windowsforms.net/articles/windowsformspainting.aspx>.

Se sua exibição inclui animações ou muda um elemento de exibição frequentemente, você deve usar o buffering duplo ou o múltiplo para preparar a imagem seguinte enquanto a atual é pintada. A enumeração de **ControlStyles** no namespace **System.Windows.Forms** aplica-se a muitos controles,



e o membro do **DoubleBuffer** pode ajudar impedir hesitações. Ligar o estilo **DoubleBuffer** fará com que seus controles de pintura sejam feitos em um buffer off-screen e então pintados todos de uma só vez na tela. Enquanto isso ajuda a impedir hesitações, ele também usa mais memória para o buffer alocado.

## Ajuste e Diagnóstico de Performance

Cuidar de assuntos de performance nos estágios de arquitetura e implementação é a maneira de custo mais efetivo para encontrar os objetivos de performance da sua aplicação. Entretanto, você só pode ser verdadeiramente eficaz na otimização da performance de suas aplicações se testar a performance de sua aplicação frequentemente e o quanto antes, na fase de desenvolvimento. Enquanto a arquitetura e testes de performance são igualmente importantes, otimizar cada componente e todo o código nestes estágios iniciais não é um uso eficiente dos recursos, e deve então ser evitado.

Conseqüentemente, sua aplicação pode sofrer de problemas de performance inesperados, que você não antecipou no estágio de arquitetura. Por exemplo, você pode experimentar problemas de performance devido à inesperada interação entre dois sistemas ou componentes, ou você pode usar códigos pré-existentes que não funcionam como esperado. Neste caso, você necessita localizar a fonte do problema de performance para que você possa endereçá-lo corretamente.

Esta seção discute um grande número ferramentas e de técnicas que lhe ajudarão a diagnosticar questões de performance, e a ajustar sua aplicação para uma performance melhor.

### Ajuste de Objetivos de Performance

Enquanto você projeta e arquiteta sua aplicação Smart Client, você deve considerar cuidadosamente as exigências em termos de performance, e definir objetivos claros para ela. Definidos esses objetivos, considere como você medirá a performance real da aplicação. Suas métricas de performance devem representar claramente as características de desempenho importantes da aplicação. Tente evitar objetivos ambíguos ou incompletos que não possam ser medidos exatamente, tais como "a aplicação deve funcionar rapidamente" ou "a aplicação deve carregar rapidamente." Você precisa saber os objetivos de performance e escalabilidade de sua aplicação de modo que você possa projetá-la a ir de encontro com esses objetivos e planejar seus teste da melhor maneira. Esteja certo de que seus objetivos sejam mensuráveis e verificáveis.

Métricas de performance bem definidas permitem que você siga o desempenho de sua aplicação exatamente, de modo que você possa determinar se a aplicação vai de encontro a seus objetivos da performance ou não. Estas métricas devem ser incluídas na planta de teste da sua aplicação, assim podem ser medidos durante a fase de testes de sua aplicação. Esta seção focaliza a definição dos objetivos específicos de performance relevantes a uma aplicação Smart Client. Se você também for projetar e construir os serviços de rede que a aplicação do cliente consumirá, você precisa definir objetivos apropriados de performance para estes também. Neste caso, você deve estar certo de considerar as exigências de desempenho do sistema como um todo e como a performance de cada parte da aplicação se relaciona com as outras peças e com o sistema em sua totalidade.

### Considerando a Perspectiva do Usuário

Conforme você determina os objetivos apropriados de performance para uma aplicação Smart Client, você deve considerar cuidadosamente a perspectiva do usuário. Para uma aplicação Smart Client, o desempenho está relacionado a usabilidade e a percepção do usuário. Para o exemplo, uma operação longa poderia ser aceitável ao usuário enquanto esse usuário puder manter-se trabalhando e for suprido com o gabarito adequado sobre o progresso da operação.

Ao determinar exigências, é frequentemente útil quebrar a funcionalidade de uma aplicação em um número de cenários do uso ou casos de utilização. Você deve identificar os exemplos e os cenários de

uso que são críticos e necessitam ir de encontro com objetivos específicos de desempenho. As tarefas que são comuns a muitos casos de uso e que são executadas frequentemente devem ser projetadas de modo que possam executar-se bem. Similarmente, tarefas de que exigem a atenção completa do usuário e que não permitem a troca para execução de outras tarefas, necessitam fornecer uma experiência eficiente e otimizada ao usuário. As tarefas que não são usadas muito frequentemente ou que não impedem que o usuário execute outras tarefas não pode necessitar ser altamente ajustado.

Para cada tarefa sensível de desempenho que você identifica, você deve precisamente definir o que o usuário faz e como a aplicação responde. Você deve também determinar que rede e recursos ou componentes do cliente cada tarefa usa. Esta informação influenciará os objetivos de performance e dirigirá os testes que medem essa performance.

Os estudos da usabilidade fornecem uma fonte muito valiosa de informação e podem influenciar extremamente a definição de objetivos de performance. Um estudo formal da usabilidade pode ser muito útil para determinar como os usuários executam seu trabalho, quais cenários do uso são comuns e quais não são, ou que tarefas os usuários executam frequentemente, e que características da aplicação são importantes em uma perspectiva de performance. Se você estiver construindo uma aplicação nova, você deve considerar fornecer um protótipo ou um modelo da aplicação para permitir um teste rudimentar de usabilidade a ser feito.

### **Considerando o Ambiente de Operação da Aplicação**

É importante avaliar o ambiente em que sua aplicação operará, já que isso pode impor barreiras a sua aplicação que devem refletir-se nos objetivos de performance que você configurar.

Os serviços de rede podem impor a performance em sua aplicação. Por exemplo, você pode ser requerido para interagir com um Web service que você não tenha nenhum controle. Em tais casos, é importante determinar a performance do serviço e determinar se este terá um efeito no desempenho de sua aplicação de cliente.

Você deve determinar também como a performance de qualquer serviço dependente e componentes podem variar com tempo. Alguns sistemas experimentaram o uso razoavelmente constante quando o outro experimenta o uso descontroladamente flutuante em determinadas horas do dia ou da semana. Estas diferenças podem afetar adversamente o desempenho de sua aplicação em horas críticas. Por exemplo, um serviço que forneça serviços da distribuição e atualização da aplicação pode ser lento para responder na manhã de segunda-feira às 9:00 AM conforme todos os usuários se atualizem à mais recente versão da aplicação.

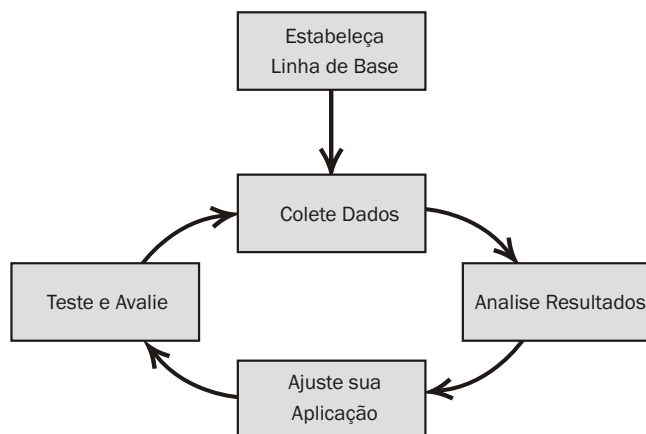
Também é importante modelar exatamente a performance de todos os sistemas e componentes dependentes, de modo que sua aplicação possa ser testada em um ambiente que imite o ambiente real em que será implementada. Para cada sistema, você deve determinar o perfil de performance e as características mínimas, médias e de pico da performance. Você pode então usar estes dados apropriadamente ao definir as exigências de desempenho para sua aplicação.

Você também deve considerar cuidadosamente o hardware em que sua aplicação funcionará. Você necessitará determinar a configuração de hardware alvo, em termos de processador, de memória, de potencialidade dos gráficos, e assim por diante — ou ao menos de uma configuração mínima abaixo de que você não pode garantir o desempenho.

Frequentemente o ambiente de negócio em que sua aplicação operará ditará algumas das exigências de desempenho mais exigentes. Por exemplo, uma aplicação que execute stock trading em tempo real será requerida para executar estas operações e para indicar todos os dados relevantes de maneira oportuna.

## Processo de Ajuste de Performance

O ajuste de performance de sua aplicação é um processo iterativo. Este processo consiste em um número de estágios que são repetidos até que a aplicação vá de encontro a seus objetivos do desempenho. (Ver Figura 8.2.)



**Figure 8.2**

*Processo de ajuste de desempenho*

Como a figura 8.2 ilustra, ajustar a performance requer que você termine os seguintes processos:

- **Estabeleça uma Linha de Base.** Antes que você comece a ajustar sua aplicação para a performance, você deve ter uma linha de base bem definida para os objetivos e métricas de performance. Isto poderia incluir especificidades tais como o tamanho do working set da aplicação, a hora de carregar dados (para o exemplo, um catálogo), a duração da transação e assim por diante.
- **Colete Dados.** Você necessitará graduar a performance da sua aplicação comparando-a aos objetivos de desempenho que você definiu. Os objetivos do desempenho devem expressar as métricas específicas e mensuráveis que permitam que você quantifique a performance de sua aplicação a qualquer momento. Para permitir que você colete dados de performance, você pode ter que instrumentar sua aplicação para que os dados requeridos de performance possam ser coletados e publicados. Algumas das opções que você tem para realizar isso são discutidas em detalhes na seção seguinte.
- **Analise Resultados.** Depois que você coletou dados de performance de sua aplicação, você poderá dar prioridade a seus esforços para o ajuste de performance determinando que características da aplicação requerem mais atenção. Adicionalmente, você pode usar estes dados para determinar onde todos os bottlenecks de performance estão. Frequentemente, você poderá somente determinar a posição exata do bottleneck recolhendo dados mais detalhados da performance: por exemplo, usando a instrumentação da aplicação. O desempenho que perfila ferramentas pode ajudar-lhe identificar o bottleneck.
- **Ajuste sua aplicação.** Depois que você identificou um bottleneck, você necessitará provavelmente modificar a aplicação ou sua configuração para tentar e resolver o problema. Você deve tentar minimizar mudanças de modo que você possa determinar o efeito das mudanças na performance da aplicação. Se você fizer mais de uma mudança ao mesmo tempo, pode ser difícil determinar que efeito cada mudança terá na performance total da aplicação.

- **Teste e Avalie.** Depois de mudar sua aplicação ou sua configuração, você deve testá-la outra vez para determinar que efeito suas mudanças têm e para permitir que os dados novos de performance sejam recolhidos. O trabalho de performance requer frequentemente mudanças arquiteturais ou outras de alto impacto, assim que testar completamente é crítico. O plano de teste da sua aplicação deve exercitar a toda a funcionalidade dos instrumentos da aplicação, porque todos os cenários antecipados e nas máquinas do cliente configuradas com hardware e o software apropriados. Se sua aplicação usar recursos de rede, você deve carregar estes recursos para que você possa obter medidas exatas de como sua aplicação funciona em tal ambiente.

O processo acima permitirá que você foque em problemas específicos de performance, medindo o desempenho total das aplicações de modo a irem de encontro aos objetivos específicos.

## Ferramentas de Performance

Há um número das ferramentas disponíveis que podem ajudar você a coletar e analisar dados da performance de sua aplicação. Cada uma das ferramentas descritas nesta seção tem funcionalidade distinta, que você pode usar para medir, analisar e encontrar bottlenecks de performance em sua aplicação.

---

**Nota:** Além das ferramentas descritas aqui, há um grande número de outras opções e ferramentas disponíveis. Para uma descrição dessas outras opções da gerenciamento de exceção e logging, veja: Exception Management Architecture Guide, at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/exceptdotnet.asp>.

---

Você deve considerar cuidadosamente as exigências exatas antes de decidir-se sobre que ferramentas são mais apropriadas as suas necessidades.

## Usando Logs e Alertas de Performance

Os Logs e os Alertas de performance são ferramentas administrativas de monitoramento da performance que funciona como parte do sistema operativo Windows. Confia nos contadores de performance que são publicados pelos vários componentes, subsistemas, e aplicações do Windows para permitir que você siga o uso do recurso e os trace graficamente contra o tempo.

Você pode usar Logs e Alertas de performance para monitorar contadores padrão, tais como o uso da memória ou o uso do processador, ou você pode definir seus próprios contadores feitos sob encomenda para monitorar uma atividade específica da aplicação.

O .NET CLR fornece um número de contadores úteis do desempenho que podem lhe dar a introspecção em como bom sua aplicação está executando. Alguns dos objetos mais relevantes para a performance são:

- **.NET CLR Memory.** Fornece dados do uso da memória de uma aplicação .NET gerenciada, incluindo a quantidade de memória que sua aplicação está usando e o tempo gasto na coleta de lixo de objetos não utilizados.
- **.NET CLR Loading.** Fornece dados sobre o número de classes e os domínios de aplicação que sua aplicação está usando e a taxa em que estão sendo carregados e descarregados.
- **.NET CLR Locks and Threads.** Fornece os dados de performance relacionados aos threads usados dentro de sua aplicação, incluindo o número de threads e a taxa da disputa entre os threads que tentam começar o acesso simultâneo a um recurso protegido.
- **.NET CLR Networking.** Fornece os contadores de performance que se relacionados a envio e ao recebimento de dados na rede, incluindo o número dos bytes enviados e recebidos por segundo e o número de conexões ativas.
- **.NET CLR Exceptions.** Fornece relatórios sobre o número de exceções que estão sendo lançadas e capturadas por sua aplicação.

Para aprender mais sobre estes contadores, seus pontos iniciais, o que medir e como medir veja a seção, "CLR and Managed Code" no Capítulo 15, "Measuring .NET Application Performance" do Improving .NET Application Performance and Scalability, em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt15.asp>.

Sua aplicação pode também fornecer contadores de performance específicos que você pode monitorar facilmente usando logs e alertas da performance. Você pode definir um contador padrão de performance, como mostrado no seguinte exemplo:

**[C#]**

```
PerformanceCounter counter = new PerformanceCounter( "Category",  
    "CounterName", false );
```

**[Visual Basic .NET]**

```
Dim counter As New PerformanceCounter("Category", "CounterName", False)
```

Uma vez que o objeto do contador de performance é criado, você pode especificar uma categoria para seus contadores de performance padrão e manter todos os contadores relacionados junto. A classe de **PerformanceCounter** é definida no namespace **System.Diagnostics**, juntamente com um grande número de outras classes que você pode usar para ler e definir contadores e categorias de performance. Para mais informação sobre como criar contadores de performance padrão veja, Knowledge Base article 317679, "How to create and make changes to a custom counter for the Windows Performance Monitor by using Visual Basic .NET," em <http://support.microsoft.com/default.aspx?scid=kb;en-us;317679>.

**Nota:** Para registrar um contador de performance, você deve primeiramente registrar a categoria. Você deve ter permissões suficientes para registrar uma categoria de contador de performance, o que pode afetar como você necessita implementar sua aplicação.

## Instrumentação

Há um grande número de ferramentas e de tecnologias que você pode se usar para ajudar instrumentar sua aplicação e gerar a informação necessária para medir a performance da aplicação. Estas ferramentas e tecnologias incluem:

- **Event Tracing for Windows (ETW).** Este sub-sistema de ETW oferece visão geral de um sistema baixo (em comparação aos logs e aos alertas do desempenho) de monitoramento de performance do sistema sob carga. Isto é primeiramente para as aplicações de usuário que devem freqüentemente registrar eventos, erros, avisos, ou exames. Para mais informação, veja "Event Tracing" no the Microsoft Platform SDK em [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/event\\_tracing.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/event_tracing.asp).
- **Enterprise Instrumentation Framework (EIF).** O EIF é uma estrutura extensível e configurável que você possa usar para instrumentar sua aplicação Smart Client. Fornece um esquema extensível do evento e um API unificado que usa os eventos existentes, logging, e seguindo os mecanismos construídos em Windows, incluindo o Windows Management Instrumentation (WMI), no Windows Event Log, e Windows Event Tracings. Simplifica extremamente a codificação requerido para publicar eventos da aplicação. Se você estiver planejando usar o EIF, você necessita instalar o EIF no computador do cliente usando o EIF.msi. Se você quiser usar o EIF em sua aplicação Smart Client, você necessita considerar esta exigência durante a implementação de sua aplicação. Para mais informação, veja "How To: Use EIF" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenethowto14.asp>.
- **Logging Application Block.** O Logging Application Block fornece componentes extensíveis e reutilizáveis do código para ajudar-lhe produzir aplicações instrumentadas. Ele constrói nas potencialidades do EIF para fornecer assim funcionalidades tais como realces ao esquema do

evento, níveis múltiplos de registro, dissipadores adicionais do evento, e assim por diante. Para mais informação, veja “Logging Application Block” em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Logging.asp>.

- **Windows Management Instrumentation (WMI).** O componente WMI é parte do sistema operativo Windows e fornece interfaces de programação para o gerenciamento de informações e o controle de acessos em uma empresa. Isto é usado geralmente por administradores de sistema para automatizar tarefas da administração usando os certificados que invocam o componente WMI. Para mais informação, veja Windows Management Instrumentation at [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi\\_start\\_page.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_start_page.asp).
- **Debug and Trace Classes.** O .NET framework fornece classes **Debug** e **Trace** sob o **System.Diagnostics** para instrumentar seu código. A classe **Debug** é usada primariamente imprimindo debugs da informação e verificando se há afirmações. A classe **Trace** permite que você instrumente configurações de liberação para monitorar a saúde de sua aplicação no momento da execução. No Visual Studio .NET, o **Trace** é padrão. Ao usar a comando-linha construir você deve adicionar a bandeira **/d:Trace** para o compilador ou **#define TRACE** no código fonte Visual C# .NET para habilitá-lo. Para códigos fonte no Visual Basic .NET, você deve adicionar **/d:TRACE=True** para o compilador da linha de comando. Para mais informações, ver Knowledge Base article 815788, “HOW TO: Trace and Debug in Visual C# .NET,” em <http://support.microsoft.com/default.aspx?scid=kb;en-us;815788>.

## CLR Profiler

O CLR Profiler é uma ferramenta que perfila a memória fornecida pela Microsoft é disponível para o download no MSDN. Permite que você olhe o heap gerenciado no processo de sua aplicação e de investigar o comportamento do coletor do lixo. Usando esta ferramenta, você pode obter informações úteis sobre a execução, o alocamento de memória, e o consumo da memória de sua aplicação. Esta informação pode ajudar-lhe compreender como sua aplicação está usando a memória e como você pode otimizar seu uso.

O CLR Profiler está disponível em <http://msdn.microsoft.com/netframework/downloads/tools/default.aspx>. Veja também “How to use CLR Profiler” at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenethowto13.asp?frame=true> para mais detalhes de como usar a ferramenta CLR Profiler.

O CLR Profiler registra informação do consumo da memória e do comportamento do coletor de lixo em um arquivo de registro. Você pode então analisar estes dados com o CLR Profiler usando o número de vistas gráficas diferentes. Algumas das vistas mais importantes são:

- **Allocation Graph.** Mostra a pilha de chamados como os objetos foram alocados. Você pode usar isso para ver o custo de cada alocamento pelo método, pelos alocamentos isolados que você não esperava, e visualizar alocamentos excessivos possíveis por um método.
- **Assembly, Module, Function, e Class Graph.** Mostra que métodos causaram o carregamento dos assemblies, as funções, os módulos, ou as classes.
- **Call Graph.** Mostra que métodos chamam outros métodos e com que frequência. Você pode usar este gráfico para determinar o custo dos chamados da biblioteca e que métodos são chamados ou como muitos chamados são feitos a um método específico.
- **Time Line.** Fornece uma visão baseada em texto, cronológica e hierárquica da execução da sua aplicação. Use esta visualização para ver que tipos são alocados e seu tamanho. Você pode também usar esta visualização para ver que assemblies são carregados como resultado de chamados a métodos e para analisar os alocamentos que você não esperava. Você pode analisar o uso dos finalizadores para identificar os métodos onde **Close** ou **Dispose** não tenham sido executados ou chamados, causando bottlenecks.

Você pode usar o CLR Profiler.exe para identificar e isolar os problemas relacionados ao coletor de lixo. Isso inclui edições do consumo da memória tais como alocações excessivas ou desconhecidas, escapes da memória, objetos long-lived, e a porcentagem do tempo gasto executando a coleta do lixo.

---

**Nota:** Para informações mais detalhadas sobre o uso da ferramenta CLR Profiler, veja "Improving .NET Application Performance and Scalability" em <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenthowto13.asp?frame=true>.

---

## Sumário

Para realizar inteiramente o potencial de uma aplicação Smart Client, você necessita considerar com cuidado questões de performance durante a fase do projeto da aplicação. Dirigindo-se a estas edições em um estágio adiantado, você pode conter custos durante o processo de projeto da aplicação e reduzir a probabilidade de ter problemas de desempenho mais tarde no ciclo de desenvolvimento.

Esse capítulo examinou as diferentes técnicas que você pode usar conforme arquiteta e projeta suas aplicações Smart Client para assegurar que você otimize seu desempenho. Mostrou também um grande número de ferramentas e técnicas que você pode usar para determinar problemas de performance dentro de suas aplicações Smart Client.

## Referências

Para mais informações, veja os seguintes links:

- <http://msdn.microsoft.com/perf>
- <http://www.windowsforms.net/Default.aspx>
- <http://msdn.microsoft.com/vstudio/using/understand/perf/>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/netcfimproveformloadperf.asp>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/highperfmanagedapps.asp>
- <http://msdn.microsoft.com/msdnmag/issues/02/08/AdvancedBasics/default.aspx>
- <http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/04/01/NET/toc.asp?frame=true>
- <http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/03/02/Multithreading/toc.asp?frame=true>

## Colaboradores e Revisores

- **Agradecimentos especiais para nossos revisores:** Mark Boulter, Jamie Cool, Keith Yedlin, Richard Turner; Ivan Medvedev; Ram Singh, Philip Vaughn; Andy Dunn, Devendra Tiwari, Eric Leonard, Ken Perilman, Per Vonge Nielsen, Naveen Yajaman, e Chris Sells
- **Agradecimentos a nossos editores e produtores pela ajuda para garantir a qualidade da experiência vivida por nossos leitores:** Sharon Smith, Microsoft; Susan Filkins, Entirenet; e Tina Burden McGrayne, Entirenet; e Sanjeev Garg, Satyam Computer Services
- **Agradecimentos a nosso time de testes:** Prashant Bansode and Guru Shankar Sundaram, InfoSys Technologies Limited
- **Agradecimentos a nossos gerenciadores de produtos:** Eugenio Pace, Microsoft; and Vasu Vijay, Electronic Data Systems



# patterns & practices



## Sobre o patterns & practices Microsoft

Os guias de patterns & practices (padrões e práticas) Microsoft contêm as recomendações específicas que ilustram como projetar, construir, implementar e operar soluções de arquitetura para negócios desafiadores e cenários técnicos. Oferecem uma orientação técnica profunda baseada na experiência real-world que vai além dos papéis brancos para ajudar os profissionais de IT da empresa, trabalhadores de informática e os desenvolvedores com a entrega de soluções rápidas.

Profissionais de IT, profissionais de informática e desenvolvedores podem escolher quatro tipos de práticas & padrões:

- **Patterns** — Os patterns (padrões) são uma maneira consistente de documentar soluções para problemas que ocorrem com frequência. Os padrões estão disponíveis para endereçar problemas de arquitetura, projeto, e execução específicos. Cada padrão tem também uma comunidade GotDotNet associada.
- **Referências de Arquitetura** — Referências de arquitetura são sistemas de nível de IT que endereçam exigências do negócio, exigências de Ciclo de Vida, e os confinamentos técnicos para cenários geralmente atuais. Referências de Arquitetura têm seu foco planejar a arquitetura em sistemas IT.
- **Referências de Blocos de Construção e Sistemas IT** — Referências de Blocos de Construção e Sistemas IT são projetos reutilizáveis de subsistema que se dirigem a desafios técnicos comuns através de uma escala larga de cenários. Muitos incluem execuções testadas da referência e aceleram o desenvolvimento.
- **Referências de Blocos de Construção e Sistemas IT** — Têm seu foco na arquitetura e implementação de subsistemas.
- **Práticas de Ciclo de Vida** — Práticas de Ciclo de Vida fornecem a orientação para tarefas fora do espaço de arquitetura e projeto tal como a distribuição e as operações em um ambiente da produção.

Os guias de patterns & practices são revistos e aprovados por equipes da engenharia de Microsoft, consultores, serviços de sustentação do produto, e por sócios e clientes. Os padrões & as guias das práticas são:

- **Comprovadas** - Baseados na experiência do campo.
- **Fidedignos** - Oferecem o melhor conselho disponível.

Para aprender mais sobre **patterns & practices** visite: <http://msdn.microsoft.com/practices>  
Para comprar guias de **patterns & practices** visite: <http://shop.microsoft.com/practices>



- **Precisos** - São validados e testados.
- **Acionáveis** - Fornecem as etapas ao sucesso.
- **Relevante** - dirigem-se aos problemas do mundo real baseados em cenários do cliente.

Os guias de patterns & practices são projetados para ajudar profissionais, trabalhadores da informação, e desenvolvedores:

#### **Reduza o custo do projeto**

- Explore os esforços de engenharia de Microsoft para conservar o tempo e o dinheiro em seus projetos.
- Siga as recomendações de Microsoft para diminuir seu risco de projeto e conseguir resultados previsíveis.

#### **Aumente a confiança nas soluções**







- Construa suas soluções em recomendações comprovadas pela Microsoft, assim você pode ter a confiança total de seus resultados.
- Confie na orientação completamente testada e suportada, nas recomendações da qualidade de produção e codifique, não simplesmente amostras.

#### **Entregue vantagens de IT estratégicas**






- Resolva seus problemas hoje e faça um exame das vantagens das tecnologias futuras da Microsoft com práticos conselhos.

**patterns & practices: Títulos Atuais**

Outubro de 2003

<b>Título</b>	<b>Versão Online</b>	<b>Livro</b>
<b>Patterns</b>		
Enterprise Solution Patterns using Microsoft .NET	<a href="http://msdn.microsoft.com/practices/type/Patterns/Enterprise/default.asp">http://msdn.microsoft.com/practices/type/Patterns/Enterprise/default.asp</a>	
Microsoft Data Patterns	<a href="http://msdn.microsoft.com/practices/type/Patterns/Data/default.asp">http://msdn.microsoft.com/practices/type/Patterns/Data/default.asp</a>	
<b>Referências de Arquitetura</b>		
Application Architecture for .NET: Designing Applications and Services	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp</a>	
Enterprise Notification Reference Architecture for Exchange 2000 Server	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnentdevgen/html/enraelp.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnentdevgen/html/enraelp.asp</a>	
Improving Web Application Security: Threats and Countermeasures	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp</a>	
Microsoft Accelerator for Six Sigma	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/sixsigma/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/sixsigma/default.asp</a>	
Microsoft Active Directory Branch Office Guide: Volume 1: Planning	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp</a>	
Microsoft Active Directory Branch Office Series Volume 2: Deployment and Operations	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp</a>	
Microsoft Content Integration Pack for Content Management Server 2001 and SharePoint Portal Server 2001	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncip/html/cip.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncip/html/cip.asp</a>	
Microsoft Exchange 2000 Server Hosting Series Volume 1: Planning	Online Version not available	
Microsoft Exchange 2000 Server Hosting Series Volume 2: Deployment	Online Version not available	







Para aprender mais sobre **patterns & practices** visite: <http://msdn.microsoft.com/practices>  
Para comprar guias de **patterns & practices** visite: <http://shop.microsoft.com/practices>

<b>Título</b>	<b>Versão Online</b>	<b>Livro</b>
Microsoft Exchange 2000 Server Upgrade Series Volume 1: Planning	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp</a>	
Microsoft Exchange 2000 Server Upgrade Series Volume 2: Deployment	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp</a>	
Microsoft Solution for Intranets	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/msi/Default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/msi/Default.asp</a>	
Microsoft Solution for Securing Wireless LANs	<a href="http://www.microsoft.com/downloads/details.aspx?FamilyId=CDB639B3-010B-47E7-B234-A27CDA291DAD&amp;displaylang=en">http://www.microsoft.com/downloads/details.aspx?FamilyId=CDB639B3-010B-47E7-B234-A27CDA291DAD&amp;displaylang=en</a>	
Microsoft Systems Architecture—Enterprise Data Center	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/edc/Default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/edc/Default.asp</a>	
Microsoft Systems Architecture—Internet Data Center	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/idc/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/idc/default.asp</a>	
The Enterprise Project Management Solution	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/epm/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/epm/default.asp</a>	
UNIX Application Migration Guide	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/ucmglp.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/ucmglp.asp</a>	
<b>Reference Building Blocks and IT Services</b>		
.NET Data Access Architecture Guide	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp</a>	
Application Updater Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp</a>	
Asynchronous Invocation Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/paiblock.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/paiblock.asp</a>	
Authentication in ASP.NET: .NET Security Guidance	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/authaspdotnet.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/authaspdotnet.asp</a>	
Building Interoperable Web Services: WS-I Basic Profile 1.0	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp</a>	
Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp</a>	

Para aprender mais sobre **patterns & practices** visite: <http://msdn.microsoft.com/practices>  
 Para comprar guias de **patterns & practices** visite: <http://shop.microsoft.com/practices>

<b>Título</b>	<b>Versão Online</b>	<b>Livro</b>
Caching Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Cachingblock.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Cachingblock.asp</a>	
Caching Architecture Guide for .Net Framework Applications	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/CachingArch.asp?frame=true">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/CachingArch.asp?frame=true</a>	
Configuration Management Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cmab.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cmab.asp</a>	
Data Access Application Block for .NET	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daab-rm.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daab-rm.asp</a>	
Designing Application-Managed Authorization	<a href="http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/damaz.asp">http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/damaz.asp</a>	
Designing Data Tier Components and Passing Data Through Tiers	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/BOAGag.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/BOAGag.asp</a>	
Exception Management Application Block for .NET	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp</a>	
Exception Management Architecture Guide	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/exceptdotnet.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/exceptdotnet.asp</a>	
Microsoft .NET/COM Migration and Interoperability	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cominterop.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cominterop.asp</a>	
Microsoft Windows Server 2003 Security Guide	<a href="http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2643C1-0685-4D89-B655-521EA6C7B4DB&amp;displaylang=en">http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2643C1-0685-4D89-B655-521EA6C7B4DB&amp;displaylang=en</a>	
Monitoring in .NET Distributed Application Design	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/monitordotnet.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/monitordotnet.asp</a>	
New Application Installation using Systems Management Server	<a href="http://www.microsoft.com/business/reducecosts/efficiency/manageability/application.mspix">http://www.microsoft.com/business/reducecosts/efficiency/manageability/application.mspix</a>	
Patch Management using Microsoft Systems Management Server - Operations Guide	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsms/pmsmsog.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsms/pmsmsog.asp</a>	
Patch Management Using Microsoft Software Update Services - Operations Guide	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsus/pmsusog.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsus/pmsusog.asp</a>	
Service Aggregation Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/serviceagg.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/serviceagg.asp</a>	
Service Monitoring and Control using Microsoft Operations Manager	<a href="http://www.microsoft.com/business/reducecosts/efficiency/manageability/monitoring.mspix">http://www.microsoft.com/business/reducecosts/efficiency/manageability/monitoring.mspix</a>	

Para aprender mais sobre **patterns & practices** visite: <http://msdn.microsoft.com/practices>  
Para comprar guias de **patterns & practices** visite: <http://shop.microsoft.com/practices>

Título	Versão Online	Livro
User Interface Process Application Block	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp</a>	
Web Service Façade for Legacy Applications	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp</a>	
Lifecycle Practices		
Backup and Restore for Internet Data Center	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/ittasks/maintain/backuprest/Default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/ittasks/maintain/backuprest/Default.asp</a>	
Deploying .NET Applications: Lifecycle Guide	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DALGRoadmap.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DALGRoadmap.asp</a>	
Microsoft Exchange 2000 Server Operations Guide	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/exchange/exchange2000/maintain/operate/opsguide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/exchange/exchange2000/maintain/operate/opsguide/default.asp</a>	
Microsoft SQL Server 2000 High Availability Series: Volume 1: Planning	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALPasp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALPasp</a>	
Microsoft SQL Server 2000 High Availability Series: Volume 2: Deployment	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALPasp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALPasp</a>	
Microsoft SQL Server 2000 Operations Guide	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/operate/opsguide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/operate/opsguide/default.asp</a>	
Operating .NET-Based Applications	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/maintain/opnetapp/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/maintain/opnetapp/default.asp</a>	
Production Debugging for .NET-Connected Applications	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DBGrm.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DBGrm.asp</a>	
Security Operations for Microsoft Windows 2000 Server	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/win2000/secwin2k/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/win2000/secwin2k/default.asp</a>	
Security Operations Guide for Exchange 2000 Server	<a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/mailexch/opsguide/default.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/mailexch/opsguide/default.asp</a>	
Team Development with Visual Studio .NET and Visual SourceSafe	<a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_rm.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_rm.asp</a>	



Este título está disponível como livro.

Para aprender mais sobre **patterns & practices** visite: <http://msdn.microsoft.com/practices>

Para comprar guias de **patterns & practices** visite: <http://shop.microsoft.com/practices>

