

**机器翻译**

**SDK**

© memoQ Zrt。

版权所有。  
www.memoQ.com

© 2004-2018 memoQ 翻译技术公司。

版权所有。  
www.memoQ.com

© 2004-2018 memoQ 翻译技术公司。

版权所有。  
www.memoQ.com

© 2004-2018 memoQ 翻译技术公司。

版权所有。  
www.memoQ.com

**目录**

[Machine Translation 1](file:///D:\GIT_repos\memoq-sdk\src\MemoQ.SDK.MT\Client\MT_SDK.docx#_Toc117500352)

[SDK 1](file:///D:\GIT_repos\memoq-sdk\src\MemoQ.SDK.MT\Client\MT_SDK.docx#_Toc117500353)

[Versions 3](#_Toc117500354)

[Overview 4](#_Toc117500355)

[The workflow for creating and distributing a plugin 4](#_Toc117500356)

[Creating and distributing a signed private plugin 5](#_Toc117500357)

[Creating and distributing a public MT plugin 5](#_Toc117500358)

[Recommended code exchange infrastructure 6](#_Toc117500359)

[Machine translation framework in memoQ 7](#_Toc117500360)

[Machine translation plugins 7](#_Toc117500361)

[Machine translation interfaces 7](#_Toc117500362)

[IModule interface 7](#_Toc117500363)

[ISession interface 7](#_Toc117500364)

[PluginDirectorBase class 7](#_Toc117500365)

[EngineBase class 7](#_Toc117500366)

[Optional: ISessionForStoringTranslations interface 8](#_Toc117500367)

[Optional: IPluginSettingsMigrator interface 8](#_Toc117500368)

[Supporting tags and formatting 8](#_Toc117500369)

[Space normalization around tags in MT services 9](#_Toc117500370)

[Scale special UI elements to high DPI 10](#_Toc117500371)

[Language data handling in MT plugins 10](#_Toc117500372)

[Machine translation SDK sample application 11](#_Toc117500373)

[Implementation steps of an MT plugin 11](#_Toc117500374)

[Create the new class library 11](#_Toc117500375)

[The plugin director 12](#_Toc117500376)

[IModule 12](#_Toc117500377)

[PluginDirectorBase 12](#_Toc117500378)

[The engine component 14](#_Toc117500379)

[The session for lookups component 15](#_Toc117500380)

[Lookup session with extended parameters 17](#_Toc117500381)

[The session for storing translations component 18](#_Toc117500382)

[Plugin settings 19](#_Toc117500383)

[Migrating settings 19](#_Toc117500384)

[The configuration dialog 20](#_Toc117500385)

[UI Design Guidelines 20](#_Toc117500386)

[Localization 22](#_Toc117500387)

[Implementation checklist 22](#_Toc117500388)

[Testing the sample plugin 23](#_Toc117500389)

[Testing the new plugins 25](#_Toc117500390)

[Testing in the sample application 25](#_Toc117500391)

[Testing a new plugin in memoQ 26](#_Toc117500392)

[Testing a new plugin in memoQ server 26](#_Toc117500393)

[Plugin supported by memoQ Server 26](#_Toc117500394)

[Checklist to update a plugin for memoQ 8.2 26](#_Toc117500395)

# **版本**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 修改者 | 变化 |
| 2013 年 5 月 11 日 | 1.0 | NG | 初始版本 |
| 2013 年 5 月 20 日 | 1.0 | BZ | 添加了工作流程章节，进行了一些修复 |
| 2013 年 5 月 23 日 | 1.0 | NG | 添加了接口定义，添加了分步测试指南 |
| 2013 年 7 月 0 日 | 1.01 | BZ、BÁ | 虚拟服务和虚拟插件已更改为使用 WCF 而不是 ASMX Web 服务：这使得可以从 Visual Studio Express 版本使用 SDK |
| 2016 年 9 月 14 日 | 2 | DÁ | 与 memoQ 8.0 相关的更改 |
| 2017 年 4 月 10 日 | 2.1 | DÁ | 支持自适应机器翻译，也就是翻译发送回机器翻译引擎 |
| 2017 年 7 月 14 日 | 3 | JM | 与 memoQ 8.2 相关的更改：插件设置和基类更改 |
| 2018 年 12 月 13 日 | 3.1 | GyK | 与 memoQ 9.0 相关的更改：ShowHelp  更新本地化（GetResourceString 返回值不正确）  概述：DummyMTPlugin |
| 2019 年 4 月 24 日 | 3.2 | GyK | **标签和格式支持**  目标框架更改：.NET Framework 4.7.2 |
| 2019 年 6 月 6 日 | 3.3 | CsÁ | 如果是公共插件，则选项表单上需要“帮助”按钮 |
| 2019 年 12 月 11 日 | 3.4 | GyK | *DisplayIcon*、*SmallIcon* 尺寸要求 |
| 2020 年 3 月 6 日 | 3.5 | PMA | *在 memoQ 服务器中测试新插件* |
| 2020 年 6 月 12 日 | 4.0 | PMA | 与 分段到 XML 和 HTML 转换器相关的更改 |
| 2020 年 6 月 20 日 | 4.1 | RT | 添加 UI 设计指南部分 |
| 2020 年 10 月 28 日 | 4.6 | PMA | 在 MS 翻译中的标签周围添加新的空格规范化功能 |
| 2020 年 10 月 29 日 | 4.6 | PMA | 扩大“实施清单” |
| 2020 年 12 月 7 日 | 4.7 | PMA | 将 Scale 特殊 UI 元素添加到高 DPI 部分 |
| 2020 年 12 月 7 日 | 4.7 | PMA | “UI设计指南”的扩展 |
| 2021 年 3 月 9 日 | 4.9 | PMA | 将私有 MT SDK 文档合并到公共 MT SDK 文档中。 |
| 2022 年 1 月 3 日 | 9.10 | PMA | 添加包使用信息：用户可以选择包的版本 |
| 2022 年 1 月 31 日 | 9.10 | PMA | MT SDK 将包含“无互联网连接处理”用例 |
| 2022 年 2 月 25 日 | 9.10 | PMA | 包括有关转义特殊字符的缺失信息 |
| 2022 年 5 月 11 日 | 9.12 | PMA | **机器翻译插件中的语言数据处理** |
| 2022 年 6 月 6 日 | 9.12 | PMA | 目标框架更改：.NET Framework 4.8。 |
| 2022 年 9 月 16 日 | 9.14 | PMA | 具有扩展参数的查找会话 |
| 2022 年 11 月 23 日 | 10.0 | PMA | SupportFuzzyForwarding属性介绍 |

# **概述**

memoQ Ltd. 使客户和第三方开发人员能够为 memoQ 创建机器翻译插件。本文档介绍了机器翻译 框架的基础知识，并提供了创建新插件的分步指南。

该文档描述了 memoQ 8.2 及更高版本支持的 MT SDK。要为早期 memoQ 版本开发插件，请参阅对应版本的文档。已有插件能保持兼容性 - 然而，这些插件不能在 memoQ 服务器上使用，并且它们在 memoQ 中的使用也有限制。

插件开发需要使用*C#*编程语言中的*.NET Framework 4.8*。

MT SDK有一个Visual Studio解决方案，可以通过Visual Studio 2015或更高版本打开。

memoQ Ltd. 开发了一个示范机器翻译插件，称为 DummyMTPlugin。在开发插件时，您必须使用此虚拟插件作为起点。如需了解更多信息，请参阅*机器翻译 SDK 示范应用*。请记得检查以下实施清单部分中的实施步骤。

# **创建和分发插件的工作流程**

当A公司想要创建一个新的MT插件时，首先他们需要基于MT SDK，用C#开发一个新的MT插件。当插件准备好后，CompanyA需要选择其类型。 memoQ 支持以下三种类型：

* **未签名的私有 MT 插件**：无需任何 memoQ 端代码审查，CompanyA 可以直接将其插件分发给用户 -或使其公开可供从共享文件夹或网站下载。 memoQ 用户可以简单地下载文件，并在 memoQ 和 memoQ 服务器中使用该插件。要了解有关未签名的私有 MT 插件的更多信息，请参阅“[在 memoQ 客户端中测试新插件](#_Testing_in_memoQ)部分" 和 “[在 memoQ 服务器中测试新插件](#_Testing_the_new)”。由于这些插件不需要任何 memoQ 端干预，CompanyA 不需要等待新的 memoQ 版本来使用它们的插件。
* **签名的私有 MT 插件**：在这种情况下也没有 memoQ 端代码审查。 CompanyA 向 memoQ Ltd. 发送插件的公钥。该公钥将存储在 memoQ 的代码库中。它确保 memoQ 可以加载插件，而无需在每次启动时警告用户有关未签名的 .dll 文件。有关生成公钥和对私有插件进行签名的详细信息，请参阅[“创建和分发签名的私有插件](#_Creating_and_distributing_1)”。注意：*插件本身*不会成为 memoQ 的一部分。如果 CompanyA 更改插件（即使添加恶意代码），公钥仍然有效，并且 memoQ 仍将运行该插件而不警告用户。一旦 memoQ Ltd. 收到公钥，“签名”插件本身只需很少的时间，并且代码中的更改通常会在下一个公共 memoQ 版本中实现。这通常意味着下一个维护版本，因此 CompanyA 不需要等待数月（直到下一个功能版本）才能使用签名的插件。
* **公共（内置）MT 插件**：此类插件是 memoQ 代码库的一部分，因此它们经过签名，安装后会在 memoQ 的内置 MT 服务中列出。在集成到 memoQ 之前，CompanyA 的插件将经过彻底的设计、本地化、代码审查和测试，以确保其完全符合 memoQ 的 MT SDK 并且没有潜在风险。请参阅“[创建和分发公共 MT 插件](#_Creating_and_distributing)”。从memoQ的角度来看，公共机器翻译插件需要市场和产品验证、规划、多轮审查和测试，如上所述。这意味着更长的周转时间。公共插件的维护也需要 memoQ 方面的大量工作，因此此类插件每年最多只能更新两次。

## **创建并分发签名的私有插件**

如果 CompanyA 创建一个新的 MT 插件并且他们希望 memoQ 对其进行签名，则需要执行以下步骤。工作流程的关键点是插件如何签名：

1. CompanyA 使用 MemoQ.AddinSigner.exe 为插件生成密钥对。该应用程序以 MT SDK 包形式发布。用法：  
     
   MemoQ.AddinSigner.exe -g <Plugin.Assembly.Name>  
     
   注意：请勿在 <Plugin.Assembly.Name> 中包含文件扩展名
2. 此命令将生成两个文件，<Plugin.Assembly.Name>PublicKey.xml 和 <Plugin.Assembly.Name>PrivatePublicKey.xml。  
   示例：如果插件的 .dll 是 *CompanyA.MyPlugin.dll*，则命令应为：  
     
   MemoQ.AddinSigner.exe –g CompanyA.MyPlugin

生成的文件将为 *CompanyA.MyPluginPublicKey.xml* 和 *CompanyA.MyPluginPrivatePublicKey.xml*。

1. CompanyA 使用 memoQ.AddinSigner.exe 使用私钥对程序集文件进行签名。

用法：

MemoQ.AddinSigner.exe -s <assembly\_file\_path> <private\_key\_file\_path>

这将生成一个 .kgsign 文件。

示例：如果插件的.dll为*CompanyA.MyPlugin.dll*，则已生成的私钥文件为*CompanyA.MyPluginPrivatePublicKey.xml*，并且它们都位于 *C:\Plugins* 文件夹，命令应该是：

MemoQ.AddinSigner.exe -s C:\Plugins\CompanyA.MyPlugin.dll C:\Plugins\CompanyA.MyPluginPrivatePublicKey.xml

结果将是一个名为 *CompanyA.MyPlugin.kgsign* 的文件。

1. CompanyA 向 memoQ Ltd. 发送插件的公钥文件 (<Plugin.Assembly.Name>PublicKey.xml) 。出于测试目的，他们还需要发送 .dll 和 .kgsign 文件。

memoQ Ltd. 需要此公钥来验证程序集的数字签名。这些更改将在下一个公共 memoQ 版本（通常是下一个维护版本）中提供。

1. CompanyA 现在可以使用该插件。他们甚至可以将其提供给其他用户，例如通过共享文件夹或网站。
   * 在 memoQ 客户端中：用户需要将 .kgsign 文件与插件的 .dll 文件一起复制到 <*memoQ\_install\_folder>\Addins* 文件夹中。
   * 在 memoQ 服务器中：用户需要将 .kgsign 文件与插件的 .dll 文件一起复制到 C:\Program *Files\Kilgray\MemoQ Server\Addins* 文件夹，并更改 .kgsign 文件的扩展名成 .skgsign。

注意：通过此工作流程，memoQ 不会测试插件本身的质量。该插件的开发者对质量负责。

## **创建并分发公共 MT 插件**

创建公共 MT 插件的推荐工作流程：

1. CompanyA 与 memoQ Ltd. 共享插件的源代码（请参阅下面的[推荐的代码交换架构](#_Recommended_code_exchange)部分。 ）
2. memoQ Ltd. 审查代码和设计，并测试插件的功能。如果需要，CompanyA 会根据审核结果执行修复。
3. memoQ Ltd. 编译 MT 插件的源代码，使用其私钥对生成的 .dll 进行签名，并使其成为 memoQ 安装程序的一部分。从此时起，MT 插件将随 memoQ 客户端安装程序一起分发。
4. MT 插件的源代码成为 memoQ Ltd 的 memoQ 代码库的一部分。
5. 有关客户/测试人员报告的错误的信息由 memoQ Ltd 转发给 CompanyA。CompanyA 负责修复这些错误；错误修复由 memoQ Ltd 审核。

上述工作流程需要确保插件满足 memoQ 的质量要求，并且不会危及整个产品的稳定性。

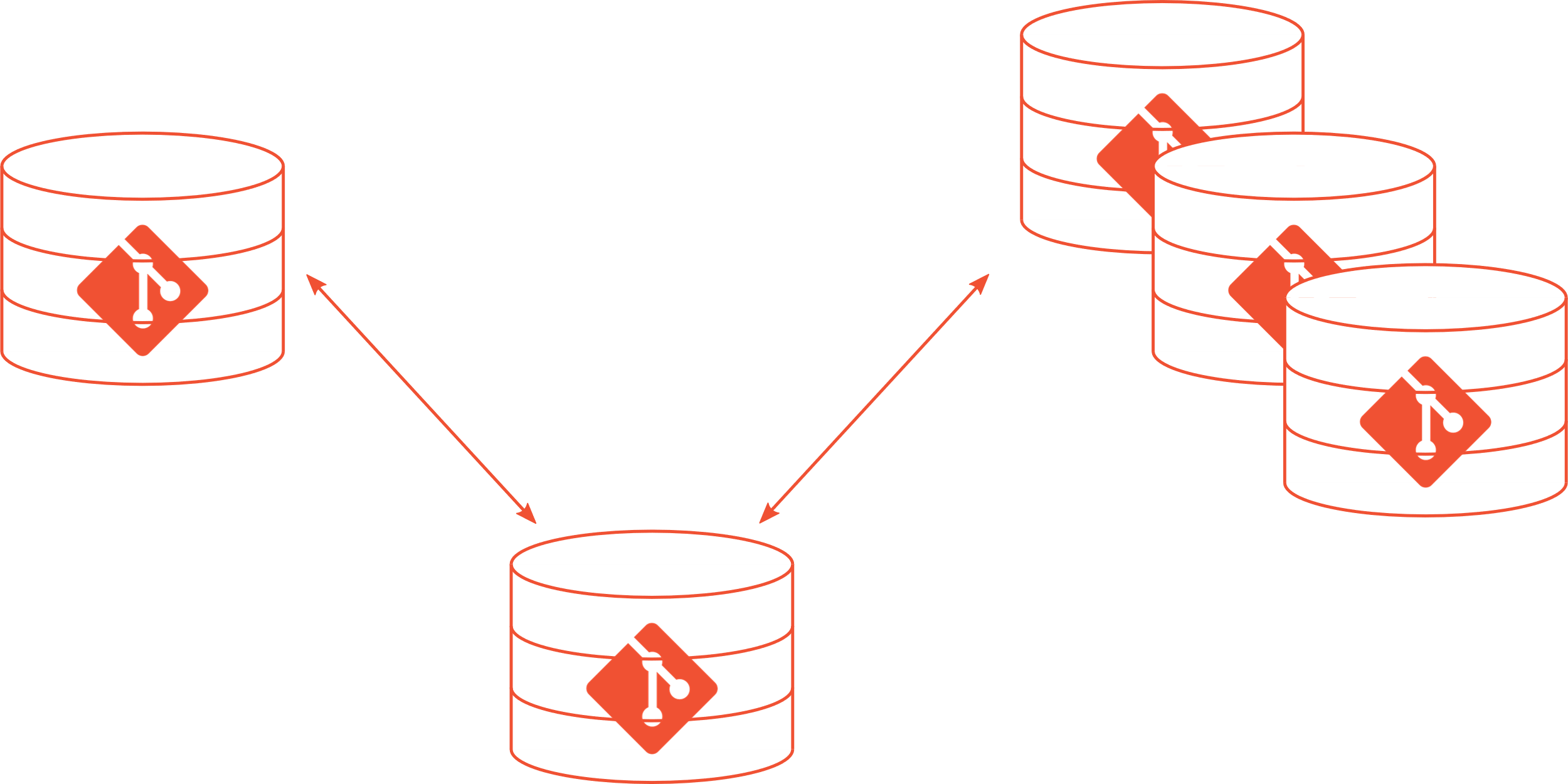
### ***推荐的代码交换架构***

插件创建者在与 memoQ Ltd 共享代码时需要遵循标准化工作流程。该工作流程基于 git 存储库。 memoQ Ltd. 提出的最低要求是代码在 git 存储库中可用，并且 memoQ 开发团队中的至少一名开发人员拥有该存储库的读取权限。该存储库的 *master* 分支需要包含插件源代码的最新版本。有时，代码将被转移到 memoQ 代码库。

这个最低要求可以通过多种方式满足，但基于与多个插件开发人员的合作，memoQ 建立了所需的工作流程（请参阅 图1如下）。工作流程中涉及三种类型的存储库 - memoQ 的代码库、交换存储库和 CompanyA 的代码库存储库（如果除了交换存储库之外还有任何存储库）。交换存储库由 CompanyA 创建和维护。实际的插件开发应该在 CompanyA 的存储库之一中进行——要么直接在交换存储库中，要么在代码库存储库之一中。

memoQ Ltd. 将使用 [git 子树集成](https://github.com/git/git/blob/master/contrib/subtree/git-subtree.txt)将插件的代码库从交换存储库纳入 memoQ 的代码库中。这需要 (1-2) 名致力于插件集成的 memoQ 开发人员对交换存储库进行读（或可能写）访问。此设置允许 memoQ Ltd. 监控插件代码的更改。如果还授予了写入权限，memoQ 开发人员可以创建小修复并将其推送到交换存储库 - 这通常比要求 CompanyA 进行细微更改更简单、更快捷。

此集成还允许 CompanyA 充分灵活地准备 MT 插件代码，以符合[实施清单](#_Implementation_checklist)。



git subtree integration

MT plugin’s exchange repository

memoQ’s code base memoQ

MT provider’s code bases

图1。推荐的代码交换架构

# **memoQ 中的机器翻译框架**

机器翻译框架允许在 memoQ 中使用外部翻译服务。 memoQ Ltd. 为 memoQ 提供了多个内置机器翻译插件（例如 Google MT 或 Microsoft MT），但公司也可以自己创建新的机器翻译插件。

## **机器翻译插件**

每个机器翻译插件都应该是一个独立的 .NET .dll，其中包含对 memoQ 代码库的以下引用：

* MemoQ.Addins.Common.dll
* MemoQ.MTInterfaces.dll

请注意，这些是应引用的唯一 memoQ 程序集。

这些库包含插件所需的所有类。 **您不得在机器翻译插件中使用任何其他外部库。**如果您认为您需要使用，请咨询memoQ Ltd。

## **机器翻译接口**

memoQ 应用程序和插件可以借助一些接口进行通信。每个机器翻译插件都应该实现这些接口：

* *MemoQ.Addins.Common.Framework.IModule*
* *MemoQ.MTInterfaces.ISession*

插件还应该派生自以下基类：

* *MemoQ.MTInterfaces.PluginDirectorBase*
* *MemoQ.MTInterfaces.EngineBase*

机器翻译插件还可以实现这些可选接口：

* *MemoQ.MTInterfaces.ISessionForStoringTranslations*
* *MemoQ.MTInterfaces.IPluginSettingsMigrator*

### ***IModule接口***

memoQ 将所有插件作为单独的模块进行管理。该接口为 memoQ 提供了一些通用函数，以便能够初始化和清理模块，并能够获取有关它们的通用信息。

### ***ISession接口***

memoQ 调用实现此接口的对象来执行查找（翻译）。一个新的会话对象是按逐个段落创建的，并且用于一次批量（翻译）操作。 ISession 对象始终由引擎对象创建。

### ***PluginDirector基类***

这是 memoQ 插件的入口点。 memoQ 在应用程序启动时为每个插件实例化一个实例，此后当 memoQ 需要与插件通信时，将使用此基类。

### ***EngineBase类***

对于特定语言组合， memoQ 在插件的director 的帮助下 请求从 EngineBase 类派生的对象。

### ***可选：ISessionForStoringTranslations 接口***

实现*ISessionForStoringTranslations*接口使插件能够存储机器翻译结果 - 如果机器翻译服务支持该行为。

### ***可选：IPluginSettingsMigrator接口***

如果您有一个在 memoQ 8.2 之前创建的机器翻译插件，并且现在您想将其升级到版本 8.2 或更高版本，或者如果您的客户拥有在 memoQ 版本 8.2 之前创建的机器翻译设置，并且他们希望在较新的 memoQ 版本中使用这些设置。

从 memoQ 8.2 开始，机器翻译插件不再存储自己的设置。相反，插件设置存储在称为“MT settings”的轻量资源中。当您升级旧版机器翻译插件以支持版本 8.2 或更高版本时，请确保实现 *IPluginSettingsMigrator* 接口，因为它允许将所有旧插件设置迁移到新的资源文件中。

## **标签和格式支持**

一些机器翻译提供商也支持标签和格式。这些提供商通常以 HTML 或 XML 格式接收它们，并在翻译文本中保留标签和格式的正确位置。 memoQ 的代码库已经提供了执行此转换的功能。您可以在 *MemoQ.Addins.Common.Utils.SegmentHtmlConverter*和*MemoQ.Addins.Common.Utils.SegmentXmlConverter中找到这些函数。*

* SegmentHtmlConverter
  + ConvertSegment2Html(Segment segment, bool includeTags, bool insertTagPlaceholder = true, Dictionary<char, string> mandatoryRepresentationOfSpecialChars = null)
  + ConvertHtml2Segment(string html, IList<InlineTag> tags)
* SegmentXmlConverter
  + ConvertSegment2Xml(Segment segment, bool includeTags, bool includeFormatting = false, bool convertPairlessTagsAsOpenClose = false)
  + ConvertXml2Segment(string xml, IList<InlineTag> tags)

*ConvertSegment2Html* 和 *ConvertSegment2Xml* 函数可让您决定是否将内联标记插入翻译的文本中。如果参数 *includeTags* 为 true，则请求将包含 memoQ 标签、内联标签和格式化标签。如果 *includeTags* 为 false 并且 *includeFormatting* 为 true，则仅包含格式标记。如果两个参数都为 false，则请求将不包含任何类型的标签。该请求将不包含 memoQ 标签的内容 - 这些标签将被唯一的占位符替换。

在 XML 转换器中，某些特殊字符不允许作为 XML 内容。这些需要转义，这需要对段进行额外的修改。此函数将特殊字符（当前为“>”、“<”和“&”）转换为 *spec\_char*  标签，其*val* 属性是原始特殊字符。例子：

**将段转换为 XML：**

“Text & Text“ → **ConvertSegment2Xml** → “Text <spec\_char val="&amp;"/> Text“

**将 XML 转换为段（以下任意一种）：**

“Text <spec\_char val="&amp;"/> Text”

“Text <spec\_char val="&"/> Text” → **ConvertXML2Segment** → “Text & Text“

“Text <spec\_char val="&#x26;"/> Text“

“Text <spec\_char val="&#38;"/> Text“

特殊字符用于标记 memoQ 内联标签的位置（*tagPlaceholders*，见下文）。引入这个特殊字符是因为提供商多次标准化（丢弃）来自转换为 html 的段的 memoQ 内联标记。对于某些提供程序，没有标记的翻译会更好，因此我们向转换器函数添加了一个可选参数来指定是否使用标记。该参数的值默认为 true。这意味着转换器将插入标记。

在 9.8 及更高版本中，两种段转换器方法还有另一个可选参数。在 ConvertSegment2Xml 中，新参数是 *convertPairlessTagsAsOpenClose*。它指示客户端是否想要根据其类型转换成对标签，或者转换为打开/关闭标签（例如，<br> 当作 <br/>)。默认情况下，该参数为 false，因此标签会根据其类型添加。 ConvertSegment2Html 方法的新参数是 *mandatoryRepresentationOfSpecialChars*。这是一个用于强制表示特殊字符的字典，由特殊字符作为key。如果客户想以特殊方式转义字符，它可能会很有用。

要将标签占位符转换回来，您需要将原始片段的内联标签列表作为参数提供给 *ConvertXml2Segment* 或 *ConvertHtml2Segment*：

var text = SegmentXMLConverter.ConvertSegment2Xml(segment, true);

var TranslationText = useTranslationService(text);

return SegmentXMLConverter.ConvertXML2Segment(translatedText, segment.ITags);

在下面的示例中，我们有一个带有格式的单词和内联标记的段。



在第一种情况下，我们需要先使用 *ConvertSegment2Xml* 方法转换源段，然后再将其发送到 MT 提供商：

This is a sample <b><i>sentence</i></b> with <inline\_tag id="0"/> an inline tag.

如果机器翻译提供商支持格式和标签，我们将收到以下响应：

Dies ist ein Beispiel <b><i>satz</i></b> mit <inline\_tag id="0"/> einem Inline-Tag.

在第二种情况下，我们需要使用 *ConvertSegment2Html* 方法转换源片段，然后再将其发送到 MT 提供商以两种不同的方式。首先，我们使用 *tagPlaceholders* 字符：

This is a sample <b>sentence</b> with <span data-mqitag="0">◿</span> an inline tag.

在第二个示例中，我们转换没有 *tagPlaceholders* 的段：

This is a sample <b>sentence</b> with <span data-mqitag="0"></span> an inline tag.

应分别使用 *ConvertXml2Segment* 或 *ConvertHtml2Segment* 方法将响应转换回（目标）分段。结果如上图右侧所示。

## **机器翻译服务中标签周围的空格标准化**

机器翻译服务有时返回的翻译在标签周围有多余的空格（或缺少空格），这需要大量的后期编辑。在版本 9.6 中，引入了一个新函数 *TagWhitespaceNormalizer.NormalizeWhitespaceAroundTags*。它允许在 memoQ 中显示之前规范 MT 引擎翻译结果中标签周围的空格。如果您认为下面详细介绍的算法可能会提高插件收到的翻译质量，则应考虑在插件的实现中使用此功能（可能取决于用户选项）。此新功能可作为 *MemoQ.Addins.Common* 中名为 *TagWhitespaceNormalizer.NormalizeWhitespaceAroundTags* 的函数来访问。

* TagWhitespaceNormalizer
  + NormalizeWhitespaceAroundTags(Segment source, Segment target, string srcLangCode, string trgLangCode)

归一化算法背后的思想是检测 MT 引擎引入的额外和不必要的空间。该算法的输入是原始源片段、提供的翻译以及源语言代码和目标语言代码。根据语言的不同，该函数的工作方式如下：

* 从非 CCJK 语言到 CCJK 语言的翻译：
* 如果打开或打开-关闭标记之前有一个单字节字符，则会在它们之间添加一个空格。
* 如果关闭或打开-关闭标记后有一个单字节字符，则会在它们之间添加一个空格。
* 它删除标签和双字节字符之间的空格。
* 从 CCJK 翻译为非 CCJK 的情况
* 标准化函数返回没有任何变化的目标片段。
* 如果语言具有匹配的 CCJK 属性
* 它返回目标片段，其标签周围的空白数量与源片段中的完全相同。

## **将特殊 UI 元素缩放至高 DPI**

在大多数情况下，UI 元素会根据显示设置（例如：缩放、方向或分辨率）动态调整大小，但某些 Windows 窗体元素（例如：按钮图标）无法执行此操作。您需要手动调整这些特殊 UI 元素的大小。从 memoQ 9.7 开始，有一个新功能可以帮助您解决此类问题。如果您有需要手动调整大小的特殊 UI 元素，请尝试在 *MemoQ.Addins.Common* 中使用以下函数：*DPIHelper.StretchImageDPI* 和 *DPIHelper.ScaleToHDPI* 。请注意，使用这些函数的插件仅适用于 memoQ 版本 9.7 或更高版本。

* DPIHelper
  + StretchImageDPI(Image imageIn, int currentDPI)
  + ScaleToHDPI(int x, int currentDPI)

当 UI 元素需要手动调整大小时，请使用这些函数。第一个输入参数是需要缩放到当前系统 DPI 值的图像 (*imageIn*) 或 UI 元素的大小（*x*，是一个整数）。第二个参数（*currentDPI*，是一个整数）是系统 DPI 值。您可以使用以下命令动态获取该值：

PropertyInfo dpiXProperty = typeof(SystemParameters).GetProperty("DpiX", BindingFlags.NonPublic | BindingFlags.Static);  
int systemDPIValue = (int)dpiXProperty.GetValue(null, null);

## **机器翻译插件中的语言数据处理**

为了使 MT 插件中的语言数据处理更加容易，MT SDK 为 MT 提供商提供了一个单元，可以有效地处理最常见的语言数据问题。目前，memoQ 的 *Kilgray.Utils* 包负责语言数据管理。

对于 MT 插件开发人员来说，最有用的功能可在公共（静态）类 *Kilgray.Utils.LanguageData* 下使用。这里最有用和最常用的函数是*GetIsoCode2LetterFromIsoCode3Letter*（将3个字母的语言代码转换为2个字母的代码）和*GetIsoCode3LetterFromIsoCode2Letter*（将2个字母的语言代码转换为3个字母）字母代码）。该类还允许查询所有 memoQ 支持的语言，或查找有关给定语言的有用信息 - 例如某种语言是否可以在 memoQ 项目中用作源语言 (*IsValidSourceLang*) 或目标语言 (*IsValidTargetLang*)，或者判断给定语言是否为 CCJK（中文 、日语、韩语）语言（*IsCCJK*）等。

但是，如果插件需要特定语言的显示名称，您应该使用该语言的代码创建一个新的 *Kilgray.Utils.Language* 对象。新创建的对象将包含显示名称。同样，主要语言代码也与 *Kilgray.Utils.Language* 类型对象关联。

*Kilgray.Utils* 是一个庞大而复杂的软件包，因此，如果您遇到困难，memoQ 很乐意为您提供帮助。

# **机器翻译 SDK 示例应用**

memoQ Ltd. 为想要实现新机器翻译插件的开发人员实现了一个小型应用程序。开发人员将能够借助此应用程序测试他们的机器翻译插件。

如果从 SDK 打开*MT\_SDK*解决方案，您可以看到三个项目：

* DummyMTPlugin
* DummyMTService
* TestClient

示例应用程序在 TestClient 项目内实现。该项目引用了 DummyMTPlugin 项目，其中包含示例机器翻译插件的实现。在开发插件时，您必须使用此虚拟插件作为起点。

第三个项目包含一个简单的 Web 服务，供示例插件使用。

在下一个部分中，我们将了解如何实现全新的机器翻译插件。

# **MT插件的实现步骤**

## **创建新的类库**

如上所述，所有插件都应作为独立库实现。为此，请创建一个面向 .NET 4.8 的新 Visual Studio *类库* 项目。然后使用 *MemoQ.Addins.Common.Framework*.*ModuleAttribute* 属性标记程序集。打开项目的 *AssemblyInfo.cs* 文件，并在最后一行后插入以下行（根据需要更改模块的名称和插件 director 类）：

[assembly: Module(ModuleName = "Dummy MT", ClassName = "DummyMTPlugin.DummyMTPluginDirector")]

memoQ 将在加载机器翻译程序集时检查此属性。 *ModuleName* 应该是机器翻译插件的名称，*ClassName* 应该是插件 director 的名称。

现在您需要设置 memoQ 库引用。必要的 .dll 文件位于 *References* 文件夹下。

注意：除了 memoQ 库之外，Newtonsoft.Json 是唯一允许的外部包。默认情况下，memoQ 包含允许的包的最新版本，但也可以使用包的特定版本：使用文本编辑器打开 *MemoQ.exe.config*，然后在 *runtime* 部分，将新的 *dependentAssembly* 单元添加到  *assemblyBinding* 组件中。 *assemblyIdentity* 元素包含有关程序集的标识信息，*bindingRedirect* 元素将一个程序集版本重定向到另一个程序集版本。

...  
<runtime>   
 <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">   
 <dependentAssembly>   
 <assemblyIdentity name="myAssembly" publicKeyToken="myAssemblyPublicToken" culture="neutral" />   
 <bindingRedirect oldVersion="the default assembly version" newVersion="the assembly version you want to redirect"/>  
 </dependentAssembly>   
 </assemblyBinding>   
</runtime>

## **插件入口**

该组件是插件的入口点。首先您需要在项目内创建一个新类。命名约定为：<*plugin\_name>PluginDirector.cs*

该类应实现以下接口：

* *MemoQ.Addincs.Common.Framework.IModule*

该类应派生自以下基类：

* *MemoQ.MTInterfaces.PluginDirectorBase*

### ***IModule***

该接口有两个函数和一个属性：

* *Cleanup*函数：实现插件的清理逻辑。
* *Initialize*函数：实现插件的初始化逻辑。
* *IsActivated* 属性：告知插件是否已激活。

The interface:

public interface IModule

{

bool IsActivated { get; }

void Initialize(IModuleEnvironment env);

void Cleanup();

}

*IModuleEnvironment* 接口提供有关使用插件的环境的信息，例如用于存储配置文件的目录路径。

### ***PluginDirectorBase***

该类有七个属性和三个函数：

* *BatchSupported* 属性：告知插件是否支持批量翻译（查找）。 memoQ 在预翻译操作期间使用批量翻译。
* *CopyrightText* 属性：应返回插件的版权信息。在 memoQ 列出了可用的插件时，这将显示在用户界面上。
* *DisplayIcon* 属性：应返回 MT 插件的图标。在 memoQ 列出了可用的插件时，该图像将显示在用户界面上。最小图标尺寸：128x128 像素。首选图标大小：256x256 像素。
* *Environment*属性：允许使用一些基本服务。 IEnvironment 接口的成员有：
  + *UILang* property: 应返回 memoQ 用户界面的两个字母的语言代码。
  + *ParseTMXSeg*函数：有一个字符串输入参数（TMX格式的段），并返回相关的memoQ *Segment*。
  + *PluginAvailabilityChanged* 函数：调用此函数来指示您的插件的可用性已更改。
  + *WriteTMXSegment* 函数：有一个输入参数（memoQ *Segment*），并将该段转换为 TMX 格式。请注意，段的可翻译属性的值不会写入 TMX。因此，为了保持这些信息完整，您需要在 TMX 往返后恢复原始属性值。
  + *GetResourceString*函数：有一个字符串输入参数（一个 key），并返回相关的本地化文本。
  + *BuildWordsOfSegment* 函数：在空格和单词边界上标记段。
  + *ShowHelp*功能：显示本地化的Web帮助；否则是部署（离线）的英语帮助。如果 *Environment* 属性实现了 *IEnvironment2* 接口（从 memoQ 9.0 开始支持），则存在此函数。要检查这一点，请使用

environment.GetType().GetInterface(nameof(IEnvironment2)) != null;

* *FriendlyName* 属性：应返回插件的人类可读名称。在 memoQ 列出了可用的插件时，这将显示在用户界面上。
* *InteractiveSupported* 属性：告诉插件是否支持交互式翻译。当用户在翻译网格中工作时，memoQ 会使用此信息，并且 memoQ 会尝试从机器翻译插件获取翻译命中。
* *PluginID* 属性：应返回插件的标识符。
* *StoringTranslationSupported* 属性：告知插件是否支持自适应（自学习）行为。
* SupportFuzzyForwarding 属性（从 memoQ 10.0 版本开始可用）：用于指示插件后面的 MT 服务是否可以在翻译方法中使用模糊 TM 匹配。如果启用了此功能，并且当前插件已选择在“编辑机器翻译设置”对话框的“发送最佳模糊 TM 匹配至”列表中，那么除了要翻译的源段外，插件还会将最佳可用 TM 匹配的源文本和目标文本一同发送给 MT 服务。
* *CreateEngine*函数：有两个输入参数（源和目标语言）。基于这些语言，应该实例化并返回一个机器翻译引擎。
* *IsLanguagePairSupported* 函数：返回插件是否支持语言对。这里不调用任何服务，根据保存的插件设置返回结果。
* *EditOptions* 函数：当用户开始配置机器翻译插件时，memoQ 调用此函数。应该显示插件的配置对话框。

The class:

/// <summary>

/// Base class for plugin director; implements <see cref="IPluginDirector2"/>

/// </summary>

public abstract class PluginDirectorBase : IPluginDirector2

{

public abstract bool BatchSupported { get; }

public abstract string CopyrightText { get; }

public abstract Image DisplayIcon { get; }

public abstract IEnvironment Environment { set; }

public abstract string FriendlyName { get; }

public abstract bool InteractiveSupported { get; }

public abstract string PluginID { get; }

public abstract bool StoringTranslationSupported { get; }

public virtual bool SupportFuzzyForwarding { get => false; }

public abstract IEngine2 CreateEngine(CreateEngineParams args);

public abstract bool IsLanguagePairSupported(LanguagePairSupportedParams args);

public abstract PluginSettings EditOptions(IWin32Window parentForm, PluginSettings settings);

}

## **引擎组件**

memoQ 调用插件管理器的 *CreateEngine* 函数来获取语言对的机器翻译引擎（取决于所需和支持的功能）。 memoQ 使用此引擎执行请求的操作类型（查找或存储翻译）。

引擎组件应从 *EngineBase* 类派生。命名约定为：<*plugin\_name>Engine.cs*。类成员有：

* *SmallIcon* 属性：当从此插件选择 MT 命中时，memoQ 在翻译结果下显示此图标。最小图标高度：128 像素。首选图标高度：256 像素。
* *SupportsFuzzyCorrection*属性：告诉引擎是否支持通过机器翻译（MatchPatch）调整模糊TM命中。 这意味着如果源片段有 TM 匹配，但并不完美，memoQ 会尝试通过将差异发送给 MT 提供商进行翻译来改进建议。如果您的机器翻译服务只能可靠地翻译完整的片段，而不能可靠地翻译部分片段（例如，两个单独的单词），请禁用此功能。但如果该服务擅长翻译片段部分，请启用它。如果禁用该功能，您的插件将不会出现在*编辑机器翻译设置*对话框的*设置*选项卡上的*MatchPatch*列表中。要了解有关 MatchPatch 的更多信息，请参阅[我们的文档](https://docs.memoq.com/current/en/Places/translation-results-list.html?#matchpatch).
* *SetProperty* 函数：设置引擎特定的属性，例如，主题领域。
* *CreateLookupSession* 函数：memoQ 调用此函数来执行翻译。在这里实例化并返回一个会话对象。该会话不会以多线程方式使用。
* *CreateStoreTranslationSession* 函数：如果插件支持自适应行为，memoQ 会调用此函数来存储翻译。您应该在此处实例化并返回一个会话对象。该会话不会以多线程方式使用。

The class:

/// <summary>

/// Base class for engines; implements <see cref="IEngine2"/>.

/// </summary>

public abstract class EngineBase : IEngine2

{

public abstract Image SmallIcon { get; }

public abstract bool SupportsFuzzyCorrection { get; }

public abstract void SetProperty(string name, string value);

public abstract ISession CreateLookupSession();

public abstract ISessionForStoringTranslations CreateStoreTranslationSession();

public abstract void Dispose();

}

*EngineBase*类继承自*IDisposable*接口。您还需要实现该接口，并且应该在dispose机制期间释放分配的资源。

## **session 的翻译组件**

该组件负责翻译（lookup）。命名约定为：<*plugin\_name>Session.cs*。接口成员有：

* TranslateCorrectSegment 的第一个重载：此函数有三个参数，全部为 MemoQ.Addins.Common.DataStructures.Segment 类型。第一个段是可翻译的段落，你可以使用其他两个段落进行模糊校正。另外两个段落是原始源段的源文本和最佳可用 TM 命中的翻译。只有当插件在“编辑机器翻译设置”对话框的“发送最佳模糊 TM 匹配至”列表中被选择，并且最佳可用 TM 命中达到 TM 设置的最小匹配阈值时，这些参数的值才不为 null。

该函数应返回一个 *TranslationResult* 对象。该对象的成员是：

* + *Translation*：应包含作为 *Segment* 对象的翻译。
  + *Confidence*：返回 0.0 到 1.0 之间转换的置信度。如果没有可用的置信水平，则应返回 0.0。
  + *Info*：返回有关翻译的附加信息，以呈现给用户（可以为空）。
  + *Exception*：如果翻译过程中发生异常，则将异常记录到该成员中。
* *TranslateCorrectSegment* 第二个重载：该函数的此重载也具有三个输入参数，但这些是*段数组*，而不是段。所有数组都具有相同的大小，并且该函数应返回 相同 大小的结果数组。

The interface:

/// <summary>

/// Session that perform actual translation. Created on a segment-by-segment

/// basis, or once for batch operations.

/// </summary>

public interface ISession : IDisposable

{

/// <summary>

/// Translate segment, possibly using a fuzzy TM hit for improvement

/// </summary>

TranslationResult TranslateCorrectSegment(Segment segm,

Segment tmSource, Segment tmTarget);

/// <summary>

/// Translate a batch of segments, possibly using a fuzzy TM hit for improvement

/// </summary>

TranslationResult[] TranslateCorrectSegment(Segment[] segs,

Segment[] tmSources, Segment[] tmTargets);

}

这两个函数都应该与 *Segment* 对象一起使用。使用它们的 *PlainText* 属性以 字符串 形式获取实际片段的内容，或使用此类中可用的任何公共方法。

*ISession* 接口继承自*IDisposable* 接口。您还需要实现该接口，并且应该在dispose机制期间释放分配的资源。

如果翻译过程中发生异常，您需要设置的*Exception*成员*TranslationResult* 类。您需要使用*MTException*类来包装原始异常。

The *TranslationResult* class is the following:

/// <summary>

/// One translated segment

/// </summary>

public class TranslationResult

{

/// <summary>

/// Translation

/// </summary>

public Segment Translation;

/// <summary>

/// Confidence of the translation between 0.0 and 1.0. If no

/// confidence level available, supply 0.0.

/// </summary>

public double Confidence;

/// <summary>

/// Additional info about the translation, to be presented to the user

/// (can be null)

/// </summary>

public string Info;

/// <summary>

/// If an exception occured during translation, then log the exception

/// into this member.

/// </summary>

public Exception Exception;

}

将翻译结果作为 *Segment* 对象返回。要从纯文本创建 *Segment* 对象，请使用 *MemoQ.Addins.Common.DataStructures.SegmentBuilder* 类（请参阅 *DummyMTSession 类以了解更多详细信息）。*

*MTException* 类：

[Serializable]

public class MTException : UserException

{

public MTException(string message, string englishMessage,

Exception innerException = null)

: base(message, englishMessage, innerException)

{ }

public MTException(SerializationInfo info, StreamingContext context)

: base(info, context)

{ }

}

使用第一个构造函数实例化 *MTException*。使用本地化的 text, 填充 *message* 参数非常重要，因为 memoQ 会在翻译网格下将此消息显示为查找错误。稍后请参阅本地化详细信息。

## **Lookup session 的扩展参数**

在 memoQ 9.14 及更高版本中， lookup sessions中提供了额外信息，以获得更好的翻译结果。在此版本之前，当通过 SDK 接收段时，MT 服务对段的来源一无所知。为了使 MT 提供商变得更加智能，memoQ 的 MT SDK 提供了该细分内容的附加信息（元数据）。如果 MT 服务可以利用元数据，则其插件应实现 *ISessionWithMetadata* 接口。 （注意：此接口的实现是可选的：如果 MT 服务无法使用元数据，您可以安全地忽略它。）

新添加的元数据包含2组8条信息。第一组包含用户提供的项目级元数据：项目 ID、客户端 ID，以及项目的领域和主题。第二组是段级别信息：项目 GUID、文档 ID、段 ID 和段状态。

这是新的扩展会话接口：

public interface ISessionWithMetadata : ISession

    {

        /// <summary>

        /// Translate segment, possibly using project and segment level metadata for improvement

        /// </summary>

        TranslationResult TranslateCorrectSegment(Segment segm, Segment tmSource, Segment tmTarget, MTRequestMetadata metaData);

        /// <summary>

        /// Translate a batch of segments, possibly using project and segment level metadata for improvement

        /// </summary>

        TranslationResult[] TranslateCorrectSegment(Segment[] segs, Segment[] tmSources, Segment[] tmTargets, MTRequestMetadata metaData);

    }

This is the *MTRequestMetadata* class:

public class MTRequestMetadata

   {

       /// <summary>

       ///Content of "Project" field from the project creation form

       /// </summary>

       public string ProjectID { get; set; }

       /// <summary>

       ///Content of "Client" info filed from the project creation form

       /// </summary>

       public string Client { get; set; }

       /// <summary>

       ///Project's domain

       /// </summary>

       public string Domain { get; set; }

       /// <summary>

       ///Project's subject

       /// </summary>

       public string Subject { get; set; }

       /// <summary>

       ///Id value of the document where the segments to translate are from

       /// </summary>

       public Guid DocumentID { get; set; }

       /// <summary>

       /// Project’s GUID identifier

       /// </summary>

       public Guid ProjectGuid { get; set; }

       /// <summary>

       ///Metadata of translation segments

       /// </summary>

       public List<SegmentMetadata> SegmentLevelMetadata { get; set; }

   }

public class SegmentMetadata

     {

        /// <summary>

        /// Original segment's ID

        /// </summary>

        public Guid SegmentID { get; set; }

        /// <summary>

        /// Shows the status of the segment

        /// </summary>

        public ushort SegmentStatus { get; set; }

        /// <summary>

        /// SegmentIndex indicates the source segment's index of the current metadata, in the source segment list

        /// </summary>

        public int SegmentIndex { get; set; }

    }

重要笔记！

* 在修补匹配的情况下，我们不能谈论完整的段落，只能谈论片段：在这种情况下，我们也没有真实的段落数据。这意味着在这种情况下，插件只会收到项目级别的信息、项目 GUID 和文档 ID。
* 如果段落不是来自标准的翻译文档，而是来自视图，文档 ID 将对应于视图 ID。通过这个 ID 信息，我们还希望指示段落之间的连接关系。

## **session 的储存翻译组件**

可选组件。它负责存储完成的翻译单元。

/// <summary>

/// Session that performs storing finished translations.

/// Created on a segment-by-segment basis, or once for batch operations.

/// </summary>

public interface ISessionForStoringTranslations : IDisposable

{

/// <summary>

/// Stores a finished translation unit.

/// </summary>

public StoreTranslation(TranslationUnit transunit);

/// <summary>

/// Stores a batch of finished translation units.

/// </summary>

/// <retuns>

/// The indices regarding the parameter array that were added succesfully.

/// </returns>

public[] StoreTranslation(TranstionUnit[] transunits);

}

*TranslationUnit* 类：

/// <summary>

/// Describes a translation unit to be stored by the MT plugin.

/// </summary>

public class TranslationUnit

{

/// <summary>

/// Translation

/// </summary>

public Segment Source;

/// <summary>

/// Translation

/// </summary>

public Segment Target;

}

## **插件设置**

您需要创建一个类来存储插件的设置。命名约定为：<*plugin\_name>Options.cs*。

注意：从 memoQ 8.2 开始，机器翻译插件不再管理（存储和加载）自己的设置。相反，所有与插件相关的设置都存储在 MT 设置资源中。所有插件设置必须是 XML 可序列化的，以便 memoQ 能够使用。用于存储选项的类必须符合 XML 序列化规则（公共 getter-setter 属性、无参数构造函数、避免不可序列化的数据类型，例如字典等）。

这些选项有两个不同的部分，用于存储常规设置和安全设置（例如密码）。 memoQ 确保安全 设置不存储为纯文本。为了促进此行为，请按照下列步骤操作：

* 创建一个类来存储常规的非安全设置。命名约定为：<*plugin\_name>GeneralOptions.cs*
* 创建一个类来存储安全 设置。命名约定为：<*plugin\_name>SecureOptions.cs***。**您存储在其中的所有内容都将被加密在 MT 设置资源中。此类是可选的：如果机器翻译插件没有任何敏感设置（例如：API 密钥、密码等），则可以省略此类。
* 从*MTInterfaces.PluginSettingsObject*派生原始选项类，并将常规类和安全类设置为类型参数。

在从基类派生时，插件基础设施会负责序列化设置。然而，插件允许通过在 GetSerializedSettings方法中提供自定义序列化来覆盖默认的序列化行为。

## **迁移设置**

当您更新旧版（8.2 之前）机器翻译插件时，请确保实现 *IPluginSettingsMigrator* 接口以保留旧插件设置。 memoQ 将自动调用director的*ReadSettingsFromFile*方法，您可以在其中加载现有选项并创建新的settings对象。

*IPluginSettingsMigrator* 接口：

public interface IPluginSettingsMigrator

{

PluginSettings ReadSettingsFromFile(string pluginSettingsDirectory);

}

您可以选择不实现该接口。在这种情况下，以前 memoQ 版本中的任何现有配置都不会自动迁移到 memoQ 8.2（或更高版本），memoQ 用户将需要手动配置插件。

## **配置对话框**

该插件应该有一个配置用户界面，用户可以在其中设置插件。您需要使用适当的用户界面元素创建一个对话框。该对话框将由插件管理器的 *EditOptions* 函数显示。命名约定为：<*plugin\_name>OptionsForm.cs*。要求如下：

* 该对话框应根据现有插件设置进行初始化。如果尚未保存设置，请使用默认设置初始化对话框。
* 仅当所有必需参数均正确配置时才允许保存设置。
* 如果用户修改了设置，则将修改收集到内存中，并仅在用户确定对话框时才保存它们。
* 不要从用户界面线程调用任何耗时操作。在后台线程中执行此操作。
* 配置对话框可以从专用应用程序域显示。一般来说，没有特定的操作来允许这样做，但是，在用户界面或代码中使用非标准实践可能会禁止这样做。建议进行测试。
* 所有公共 MT 插件都需要链接到正确的 memoQ 帮助页面的“帮助”按钮。
* UI 在高 DPI 设置下正确显示。
* 与安全设置相关的参数用字符“\*”（API 密钥、密码等）屏蔽。
* 在凭证数据验证或与 MT 提供商服务器的任何其他必要交互过程中，没有互联网连接或质量差可能会导致问题。 memoQ（异常处理的统一系统，包括互联网连接错误）最近的一项改进允许对公共插件的“无互联网连接”问题进行全局处理。为了在 memoQ 的统一系统中进行处理，插件抛出的异常需要是 *WebException* 类型，并且它们的状态代码需要是 *WebExceptionStatus.ConnectFailure* 或*WebExceptionStatus.NameResolutionFailure*。

## **用户界面设计指南**

如果您只记得这些准则之一，它应该是：  
  
**不要重新发明轮子！**

大多数应用程序的结构如此，是有原因的——因为它有效。为什么？因为用户已经学会了，并且已经习惯了。如果他们遇到新的用户界面 (UI) 组件或行为与其他应用程序不同的组件，他们将需要学习该新行为。这需要不必要的时间和精力。最好在产品中的各处使用经过尝试和测试的布局，并进行一些细微的调整以适应您的目标。这样，用户将能够轻松使用您的插件，而无需学习新的布局。所以：

**使用标准 Windows UI**

使用标准的Windows UI组件，并使整个插件看起来像Windows和memoQ。您的用户会感到宾至如归，并且能够轻松使用该插件。

以下是一些有关如何在 Windows 应用中创建直观的用户界面和用户体验的 Microsoft 资源：  
<https://docs.microsoft.com/en-us/windows/win32/uxguide/top-violations><https://docs.microsoft.com/en-us/windows/win32/appuistart/-user-interface-principles><https://docs.microsoft.com/en-us/windows/win32/uxguide/guidelines>

（我们知道即使 memoQ 也没有遵循所有这些资源，但我们正在努力。）

我们建议您阅读并遵循所有这些准则。但这里有一个较短的列表：

* 接近  
  就像在生活中一样，逻辑上相关的组件在物理上应该靠近在一起。但不要将组件放置得太近，它们之间始终留有一些空白空间。
* 结盟  
  物品不应随意放置：每个组件都应与另一个组件具有视觉联系。
* 尺寸  
  使相同的 UI 组件具有相同的高度，例如所有按钮的高度应相同。如果您不确定组件的尺寸，最简单的方法是在 Windows 对话框上测量。
* 字体  
  我们建议使用“Segoe UI”12 pt 作为通用字体。仅当标题较短且绝对有必要突出显示这些单词时才使用半粗体。避免斜体。所有文本都应为黑色，链接（或命令链接）除外，应为蓝色 (#0000EE)。重要的错误文本可能是红色的（#EE0000），但不要过度使用它。
* 图标和标志

如果有一个众所周知的 Windows 图标来指示某些内容（例如：黄色警告标志、蓝色信息点等），请使用这些图标，不要创建自己的图标。对于自定义图标和徽标，请使用透明背景。他们会看起来更漂亮、更专业。

* 缩放至高 DPI

UI 上的图标必须缩放。请检查 100、125、150、200 和 300 缩放比例。有关详细信息，请参阅[“将特殊 UI 元素缩放至高 DPI”](#_Scale_special_UI) 部分。

* 本地化  
  永远记住，用户界面也将本地化为其他语言。德语或西班牙语文本通常比英语长。一般来说，为文本留出 50% 的空间，以便其他语言适合。对于较短的文本（1 或 2 个单词），留出更多空间：有时，即使多出 100% 的空间也可能太短。
* 添加工具提示  
  工具提示是一种简单但功能强大的方式，可以为用户提供即时帮助。为尽可能多的 UI 组件使用工具提示。对于复杂的任务，当用户不清楚该怎么做时，可以使用众所周知的带有“i”的蓝点图标来向用户表明这里有一些帮助。
* 测试用户界面  
  将其展示给您的同事或不了解该功能的朋友。询问他们认为这些组件是如何工作的。这比根本不测试要好得多，他们的答案甚至可能会让您感到惊讶。

如有疑问，请通过 [design@memoq.com](邮箱：design@memoq.com) 询问我们。如果您需要有关插件用户界面的建议，memoQ 的设计团队很乐意为您提供帮助。

## **本地化**

第三方机器翻译插件将由 memoQ Ltd 进行本地化。 *IEnvironment* 接口为开发人员能够从机器翻译环境中获取本地化文本。

图形用户界面上出现的所有文本信息都应该本地化。因此，插件的开发人员必须向 memoQ Ltd 提供这些字符串的列表。该列表应包含键值对。键必须唯一标识字符串值。您将能够使用 *GetResorceString* 函数在插件中使用这些本地化文本 - 只需将所需文本的键传递给该函数即可。除了这个，该函数还有另一个参数，*pluginId*。该参数应该是机器翻译插件的唯一ID。将此标识符作为公共常量放入 *PluginDirector* 类中。

可能 GetResourceString 函数以“MTPlugin.<PluginId>.<Key>”的形式返回所搜索的资源键（例如：MTPlugin.MyPlugin.ErrorMsg）。在这种情况下，插件应该使用自己的默认字符串。

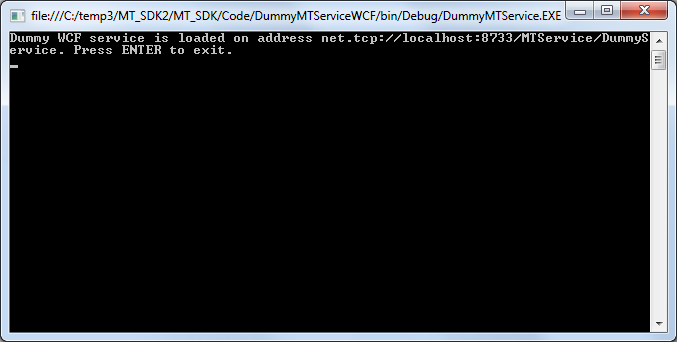
# ***实施清单***

如果您完成了机器翻译插件的实现，您需要检查：

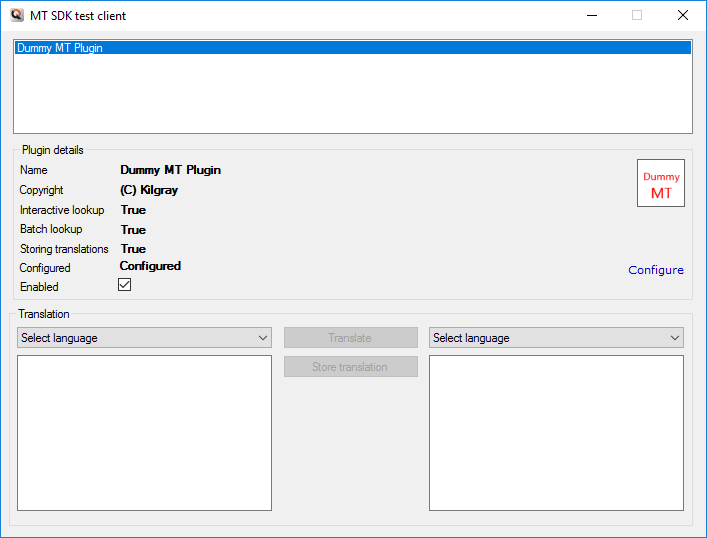
* 该实现位于单个类库中，其中包含对必要的 memoQ 库的引用。类库是用C#编写的。
* 实现过程中添加的所有源代码文本（注释、函数命名、变量、类等）都是英文的。
* 不使用任何包引用，但以下列表中的包除外：*MemoQ.Addins.Common、Kilgray.Utils、MemoQ.MTInterfaces、Newtonsoft.Json、Microsoft.IdentityModel.Tokens、System.IdentityModel.Tokens.Jwt。*
* 类库的 *AssemblyInfo.cs* 包含 *ModuleAttribute* 属性。
* 有一个插件 director 组件，它正确实现了*IModule*接口并派生自*PluginDirectorBase*类。
* 所有分配的资源都在插件 director 中正确处置。
* 有一个引擎组件正确地实现了*EngineBase*接口。
* 所有分配的资源都在引擎中正确配置。
* 有一个会话组件正确地实现了*ISession*接口。
* *MTException* 类用于包装翻译过程中发生的原始异常。
* 所有分配的资源都在会话中正确处置。
* 有一个带有适当的*generic*和*secure*子类的选项类（安全选项类可以省略）。
* options类是一个简单的实体类，不调用任何服务，只是返回保存的或默认的设置。
* options 类不会存储和加载自己的设置。
* 有一个配置对话框，用户可以在其中配置插件。
* 只有正确配置所有必填参数后，用户才能保存设置。
* 该对话框将用户的更改收集在内存中，并且仅在用户确定该对话框时才保存。
* 对话框不会在用户界面线程中调用任何阻塞服务或冗长的操作；它必须使用后台线程。
* 翻译服务仅在配置和翻译期间被调用。其他地方都使用存储的插件设置来返回插件信息（例如，插件支持的语言）。
* 所有 UI 在高 DPI 设置下都能正确显示。
* 与安全设置相关的参数（API 密钥、密码等）在设置 UI 上用星号“\*”字符屏蔽。

# **测试示例插件**

测试示例插件：在Visual Studio中打开*MT\_SDK*解决方案，并将*TestClient*和*DummyMTService*项目设置为启动项目，然后开始调试。 *DummyMTService* 作为控制台应用程序运行并模拟 MT 服务：

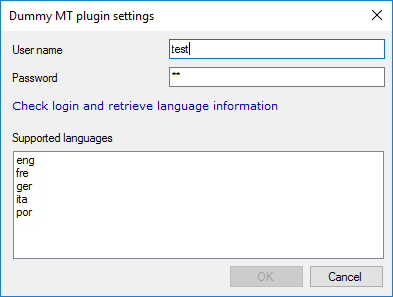


*TestClient* 模拟 memoQ – 它加载并使用 MT 插件：



目前仅注册了一个 MT 插件。虚拟插件的属性位于*插件详细信息*部分。为了能够使用该插件翻译文本，您需要先对其进行配置。点击*配置*链接。

该对话框允许您设置插件。在*用户名*字段中输入一些内容，在*密码*字段中输入其他内容，然后单击*检查登录并检索语言信息*链接。一个错误对话框出现，因为示例插件仅在用户名和密码相同时才允许登录。现在，在*用户名*和*密码*字段中输入相同的字符串，然后单击*检查登录* 再次链接。现在出现支持的语言：

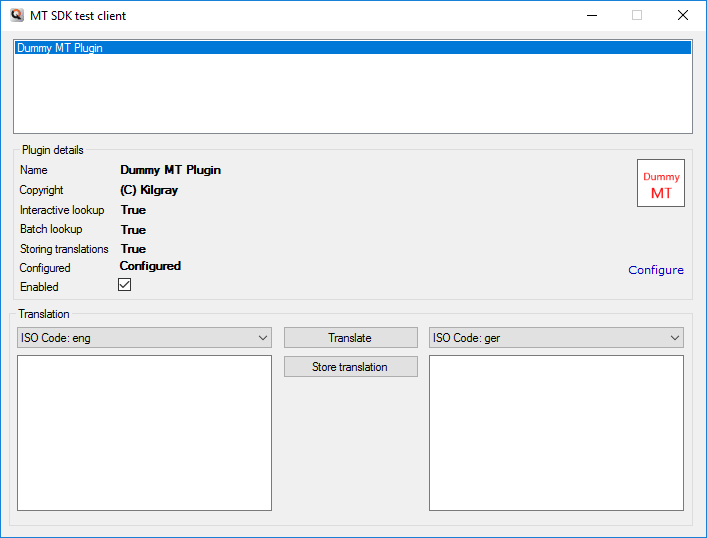


单击*确定*按钮并在*插件详细信息*部分启用插件。

要翻译某些内容，请选择语言对。如果您选择插件不支持的语言，您将收到错误消息“所选插件不支持此语言对。”

选择支持的语言对并在源文本框中输入内容，然后单击*翻译*按钮。几秒钟后翻译将出现在目标文本框中。

如果您在源文本框中输入多行，则插件会执行批量翻译。 否则， 它会进行简单的翻译。



# **测试新插件**

## **在示例应用程序中进行测试**

如果您想测试您的机器翻译插件，您需要将您的项目作为项目引用添加到 *MT\_SDK* 项目中。之后您需要扩展*MainForm*类的构造函数。在“添加其他插件控制器”注释后插入以下行（实例化您自己的插件控制器而不是*DummyMTPluginDirector*）：

plugins.Add(PluginInfoFactory.Create(new DummyMTPluginDirector()));

如果插件正确实现，它将列在示例应用程序的主窗体上。如果您从列表中选择插件，您可以在“插件详细信息”框中看到其一般信息。如果您点击*“配置”*链接，您可以设置您的插件。如果插件已配置并启用，您将能够测试翻译。选择源语言和目标语言（如果您选择不受支持的语言对，两个文本框之间会出现一条红色消息），然后在左侧文本框中输入内容。如果文本框包含多行文本，则会调用批量翻译。如果翻译过程中出现任何异常，将会出现一个消息框。

## **在 memoQ 中测试新插件**

您还可以在 memoQ（版本 7.8.55 及更高版本）中测试您的 MT 插件。 首先，将插件的 .dll 文件复制到 memoQ 安装文件夹下的 *Addins* 文件夹中。默认情况下，memoQ 需要在启动时确认才能加载未签名的插件。要自动加载插件，您需要在 *%PROGRAMDATA%/MemoQ* 文件夹中创建一个名为 *ClientDevConfig.xml* 的 XML 文件，其中包含以下内容：

<?xml version="1.0" encoding="utf-8"?>

<ClientDevConfig>

<LoadUnsignedPlugins>true</LoadUnsignedPlugins>

</ClientDevConfig>

现在，如果正确实施，memoQ 将加载您的插件。

## **在 memoQ 服务器中测试新插件**

您还可以在 memoQ 服务器中测试您的 MT 插件。首先，将插件 dll 文件复制到 memoQ 服务器安装文件夹下的 *Addins* 文件夹中 - 就像 memoQ 一样。要自动加载未签名的插件，请将 .dll 插件的文件名（不带文件扩展名）添加到 *%PROGRAMDATA%\MemoQ Server*  文件夹中的 *UserApprovedUnsignedMTplugins.xml*  文件中，然后重新启动 memoQ 服务器。

<?xml version="1.0" encoding="utf-8"?>

<ApprovedUnsignedMTPluginsCatalog xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<ApprovedUnsignedMTPlugins>

<Plugin>

<Name>MemoQ.ExampleMT</Name>

</Plugin>

<Plugin>

<Name>MemoQ.TestMT</Name>

</Plugin>

</ApprovedUnsignedMTPlugins>

</ApprovedUnsignedMTPluginsCatalog>

# ***memoQ Server 支持的插件***

memoQ 服务器 8.2 及更高版本通过 MT 设置资源支持机器翻译插件作为项目的一部分。插件架构的构建使得如果插件安装在 memoQ 服务器中，memoQ 客户端就可以使用它，而无需在本地安装该插件。这允许对插件及其设置进行集中管理和配置——用户无权访问使用 MT 服务所需的密码和其他敏感信息，但他们仍然可以执行查找。

memoQ 服务器不会加载旧版插件，只会加载符合下面清单的插件。

插件开发人员应该意识到，配置插件设置是在 memoQ 的用户界面上完成的 - 即使插件未在本地安装。为了显示配置用户界面，memoQ 从 memoQ 服务器下载插件的 dll。 （插件 dll 然后被丢弃；它永远不会写入磁盘。）如果插件是使用外部依赖项构建的，它仍然必须能够在没有这些外部依赖的情况下显示配置用户界面。

# ***更新 memoQ 8.2 插件的清单***

要使旧版（版本 8.0）插件及其代码库与 memoQ 版本 8.2 或更高版本完全兼容，请通过执行以下步骤更新库。

* 更新*IPluginDirector2*接口的实现类。
  + 不要直接使用此接口，而是从 *PluginDirectorBase* 派生。
  + 重写必要的方法和字段。
* 从director 中完全删除*IModuleEx* 实现。
* 更新*IEngine2*接口的实现类。
  + 不要直接使用此接口，而是从 *EngineBase* 派生。
  + 重写必要的方法和字段。
* 更新插件的选项类。
  + 不要使用静态字段和方法来访问选项实例。
  + 相反，在需要的地方传递选项对象。
* 创建两个新选项类：一类用于常规设置，一类用于安全 设置。 （安全设置类是可选的。）
  + 您的原始选项类应从 *PluginSettingsObject* 派生，并且您应将通用类和安全类作为类型参数传递。
  + 为选项类创建两个构造函数：一个带有 *PluginSettings* 参数，另一个带有两个常规和安全设置参数。确保将这些参数传递给基类。
  + 将现有选项字段从原始选项类移动到正确的类中。
  + 更新插件的代码以通过常规和安全设置类访问这些字段。
* 如果您希望通过将旧插件设置迁移到 MT 设置资源中来保留它们，则 director 应实现 *IPluginSettingsMigrator* 接口。