IBM Cúram Social Program Management

Cúram 人员和潜在人员证据开发人员指南

V 6.0.5

IBM Cúram Social Program Management

Cúram 人员和潜在人员证据开发人员指南

V 6.0.5

注解

在使用本资料及其支持的产品之前，请阅读第 33 页的『声明』中的信息。

修订时间: 2013 年 5 月

此修订版适用于 IBM Cúram Social Program Management V6.0 5 及其所有后续发行版，直到在新版本中另有说明为止。

目录

图

表

第 1 章 简介

1.1 目的

本指南的目的是提供对人员/潜在人员证据及其组件的高级技术了解。本指南还概述可用定制选项和扩展点，提供有关如何实现这些定制的指示信息。

1.2 受众

本指南的目标读者是想要实现人员/潜在人员证据解决方案的开发人员和架构设计师。

要点: 本指南只适用于使用具有人员和潜在人员动态证据的参与者申请的读者。

1.3 先决条件

本文档假定读者熟悉以下文档。

- 《*Cúram 证据指南*》
- 《*Cúram 参与者指南*》
- *Cúram Dynamic Evidence Configuration Guide*
- *Google Guide*

1.4 本指南中的章节

下表描述了该指南中的章节:

人员/潜在人员证据概述
本章提供人员/潜在人员证据关键技术方面的高级概述。

设计人员/潜在人员证据解决方案
本章概述在设计人员/潜在人员证据解决方案时应考虑的一些设计注意事项。

数据映射
本章描述将随应用程序提供的动态证据类型映射到旧数据库表的数据映射。

定制人员/潜在人员证据
本章描述人员/潜在人员证据的可用定制选项和扩展点。

第 2 章 人员/潜在人员证据概述

2.1 人员/潜在人员数据作为证据

存储的有关人员和潜在人员的一些信息保存为证据，可以在系统上的案例之间共享此证据。若干 Cúram 组件和技术集合以支持存储人员和潜在人员证据以及该证据在系统中的流动:

- Cúram 动态证据用于存储已捕获的人员/潜在人员数据和执行基本验证。
- Cúram Express Rules 用于对已捕获数据执行复杂验证。
- Cúram Evidence Broker 可用于代理案例之间的数据。
- Cúram 验证可用于在案例之间代理已捕获数据时应用验证。

根据不同的业务需求，可能需要一些配置和/或定制。本章高度概括了系统如何管理人员/潜在人员数据以及标识何时需要配置和/或定制。

注: 在 Cúram 实现的设计过程中应给出系统必需业务行为的注意事项。要想了解应如何配置系统以支持业务需求，应先阅读《Cúram 参与者指南》和《Cúram 证据指南》。

2.2 如何管理人员/潜在人员证据?

将人员和潜在人员数据作为证据进行管理基于以下关键基础:

- 每个人员和潜在人员都有注册时创建“在底层”的关联的人员或潜在人员案例（参与者数据案例）。
- 人员和潜在人员数据作为证据存储在动态证据表中，由与人员和/或潜在人员关联的动态证据类型提供描述。
- 作为证据记录的数据会复制回旧数据库表中，所以，旧数据库表应与动态证据同步。
- 编辑人员或潜在人员记录时，会从动态证据表检索数据，也会将数据写入动态证据表，然后再次复制回旧数据库表中。
- 在某些情况下，申请屏幕和处理仍将会从旧数据库表中进行读取。
- 可使用 Cúram Evidence Broker 将人员和潜在人员案例类型配置为具有代理到其他案例和从其他案例代理的参与者数据。

2.2.1 人员/潜在人员证据类型

提供了若干个动态证据类型，并且这些动态证据类型与人员和潜在人员参与者相关联。不得除去这些证据类型及其属性，也不得将其与人员和潜在人员取消关联。

在需要将额外数据作为人员和潜在人员证据进行管理时，可以按照 Cúram Dynamic Evidence Configuration Guide 中所述创建新的证据类型，并将其与管理组件中的人员和潜在人员关联。同样的，可以将新的属性添加至现有动态证据类型。当添加至新证据类型或现有证据类型的数据已显示在现有旧数据库表中时，必须执行额外的定制工作:

- 必须扩展将数据从动态证据表复制到旧数据库表（“replicator”）的代码，以将正在存储的附加数据复制为证据
- 当数据已存在于旧数据库表上时，必须将此数据复制到等同的动态证据表中。必须扩展执行此操作的代码（“converter”），以将附加数据转换为证据。

“定制人员/潜在人员证据”一章中提供了上述两项定制工作的更多详细信息和示例实现。

2.2.2 证据验证

人员和潜在人员证据在创建和编辑后验证。这些验证以下列两种方式之一实现:

- 使用“动态证据编辑器”验证功能。例如，必填字段验证。
- 使用 Cúram Express 规则集: 例如，跨证据验证。

添加新动态证据类型或属性时，客户应使用其中一种机制来添加任何需要的验证。Cúram Dynamic Evidence Configuration Guide 中对此进行了更详细的描述。

如果证据对人员和潜在人员进行代理，那么将不会检查这些验证。必须始终接受代理的证据以防止其丢失，因为对人员和潜在人员而言，没有“入局证据”的概念。在输入人员/潜在人员证据后，将立即对其进行验证。但将自动接受并激活其他案例中代理的证据，即便验证检查失败也如此。对于其他案例类型，如果人员/潜在人员证据在案例中代理，那么验证失败将能防止该证据被激活。

2.2.3 证据共享

证据框架提供了在人员/潜在人员、申请案例和在审案例之间共享证据的功能。证据代理工具支持并调解此证据共享。证据共享具有单向性，且因每个证据类型的不同而异。这意味着，不同目标可采用不同方式接收并共享证据类型。需要时，一种案例类型可能会接收共享证据，但是可能无法共享自身的证据。有三种主要业务功能，这些业务功能将触发证据代理工具以广播证据:

- 在新人员添加到目标案例中时。例如，在诸如标识证据等人员/潜在人员证据已配置为共享到综合案例并且有人员添加至综合案例时，证据代理工具将首先检查以确定该人员是否具有人员/潜在人员证据。如果找到证据，那么证据代理工具会接着检查有效标识证据并将它共享到综合案例。
- 在对源案例进行证据更改时。例如，当对人员的标识证据进行更改时，证据代理工具将把这些更改共享到综合案例。
- 在创建新的目标案例时。例如，每当创建新的综合案例时，证据代理工具将搜索要共享的人员/潜在人员标识证据。如果找到此证据，那么证据代理工具会将该标识证据共享到综合案例。

有关证据代理工具的更多详细信息，请参阅《Cúram 证据代理工具指南》。

第 3 章 设计人员/潜在人员证据解决方案

设计人员/潜在人员证据解决方案时，设计者应考虑数据、其结构、约束以及系统周边此数据的流量。

3.1 数据: 动态证据类型

3.1.1 结构

人员/潜在人员证据首先存储为动态证据，而代表其的数据结构即为动态证据类型。动态证据类型定义数据及其类型和行为（如易失性和已计算属性等）。定义新动态证据类型后，必须将其激活并关联相关案例类型、人员和潜在人员。有关如何定义动态证据类型的进一步信息，可在 Cúram Dynamic Evidence Configuration Guide 中找到。

注意事项

- 证据是否会随时间变化？
- 是否为相反证据类型？如果是，那么证据类型应具有参与者和相关参与者属性。
- 证据可供哪些案例类型使用？
- 如果证据类型是常用类型，那么考虑将其设置为“首选”。这将使案例工作人员能够快速创建常用记录证据类型的证据。

3.1.2 约束

3.1.2.1 验证

“动态证据编辑器”中提供若干个标准验证，这些验证通常用于证据处理。可以使用“Cúram 表达式规则”来包含诸如跨证据验证等更为复杂的验证。可在 Cúram Dynamic Evidence Configuration Guide 中找到有关验证的更多信息。

注意事项

- 确保数据完整性需要什么验证？
- 在输入人员/潜在人员证据后，将立即对其进行验证，但将自动接受并激活其他案例中代理的证据，即便验证检查失败也如此。对于其他案例类型，如果人员/潜在人员证据在案例中代理，那么验证失败将能防止该证据被激活。
- 包含强制成功约束所需的任何验证。
- 尽可能尝试使用标准验证模式。仅当无法使用标准验证实现的情况下才应开发验证规则集。
- 如果要开发验证规则集以进行更复杂的验证，请留意数据检索的执行方式。如果执行不正确，那么会对性能造成重大影响。

要点: 系统进程依赖于随申请提供的验证，不应移除或变更这些验证。

3.1.2.2 验证

本节仅适用于被许可使用验证组件的读者。

验证是检查证据准确性的过程。证据的验证可采用许多形式；可通过文档（例如，出生证明或银行对帐单）或通过口头方式（例如，电话）来提供验证。捕获证据后，将调用验证引擎以确定证据是否需要验证。

注: 除了已代理到人员/潜在人员记录的证据，其余证据都必须在符合所有必需验证要求后才能激活。

有关验证及其配置的更多信息，请参阅《Cúram 验证指南》。

注意事项

- 此证据是否需要验证？
- 验证的相关规则有哪些？
- 客户需要提供什么信息？

3.2 流: 证据代理工具

证据代理工具是允许证据在整个系统共享的机制。当证据代理工具将证据广播至人员/潜在人员记录时，人员/潜在人员记录上会自动接受并激活该证据，因此用户无需手动接受并激活记录。有关证据代理工具和配置选项的更多信息，请参阅《Cúram 证据代理工具指南》。有关建议的代理方法，请参阅《Cúram 证据指南》。

注意事项

- 对不止一种案例类型使用了同一种证据类型吗？如果是，是否应该将此证据的更改传达给其他案例？
- 是应该将目标案例设置为自动接受更改，还是应该强制案例工作者介入决定是否接受此入局证据？
- 为使系统处理能够正常运行，将“参与者管理器”外记录的人员/潜在人员数据共享回“参与者管理器”非常重要。

3.3 Cúram Express Rules: 案例资格/权利计算

应分析 Cúram Express Rules 用于从旧数据库表读取参与者数据（用于案例资格和权利计算）的区域，并且判定哪个位置应当作为此数据的信息来源。共有三个选项，这三个选项各有利弊:

- 从由参与者管理器存储的动态证据读取参与者数据
- 读取案例上已代理的参与者数据
- 继续从旧表读取

3.3.1 从由参与者管理器存储的动态证据读取参与者数据

注意事项

- 清理主要数据源
- 证据更改会导致立即重新计算
- 案例工作人员无法复审

3.3.2 读取案例上已代理的参与者数据

注意事项

- 这是用于任何新部署的建议选项
- 仅在证据激活时更改会生效
- 必须将证据类型配置为在案例上代理

3.3.3 继续从旧表读取

注意事项

- 应当仔细考虑此选项并且仅建议用于升级客户

第 4 章 动态证据类型数据映射

下表显示从动态证据类型到旧数据库表的数据映射。

注: 复制器将执行此映射，而转换器将执行此反向映射。

4.1 地址

表 1. 地址映射

动态证据属性	数据库列
participant	ConcernRoleAddress.concernRoleID（使用 caseParticipantRoleID 进行计算）
address	Address.addressData
fromDate	ConcernRoleAddress.startDate
toDate	ConcernRoleAddress.endDate
addressType	ConcernRoleAddress.typeCode
comments	ConcernRoleAddress.comments

4.2 银行帐户

表 2. 银行帐户映射

动态证据属性	数据库列
participant	ConcernRoleBankAccount.concernRoleID（使用 caseParticipantRoleID 进行计算）
accountName	BankAccount.name
accountNumber	BankAccount.accountNumber
accountType	BankAccount.typeCode
sortCode	BankAccount.bankSortCode
fromDate	BankAccount.startDate
toDate	BankAccount.endDate
accountStatus	BankAccount.bankAccountStatus
jointAccountInd	BankAccount.jointAccountInd
comments	BankAccount.comments

4.3 出生和死亡

表 3. 出生和死亡映射

动态证据属性	数据库列
person	Person/ProspectPerson.concernRoleID（使用 caseParticipantRoleID 进行计算）
birthLastName	Person/ProspectPerson.personBirthName

表 3. 出生和死亡映射 (续)

动态证据属性	数据库列
mothersBirthLastName	Person/ProspectPerson.motherBirthSurname
dateOfBirth	Person/ProspectPerson.dateOfBirth
dateOfDeath	Person/ProspectPerson.dateOfDeath
comments	不适用

4.4 联系方式首选项

表 4. 首选联系方式映射

动态证据属性	数据库列
participant	ConcernRole.concernRoleID（使用 caseParticipantRoleID 进行计算）
preferredLanguage	ConcernRole.preferredLanguage
preferredCommunication	ConcernRole.prefCommMethod
comments	不适用

4.5 电子邮件地址

表 5. 电子邮件地址映射

动态证据属性	数据库列
participant	ConcernRoleEmailAddress.concernRoleID（使用 caseParticipantRoleID 进行计算）
emailAddress	EmailAddress.emailAddress
fromDate	ConcernRoleEmailAddress.startDate
toDate	ConcernRoleEmailAddress.endDate
emailAddressType	ConcernRoleEmailAddress.typeCode
comments	EmailAddress.comments

4.6 性别

表 6. 性别映射

动态证据属性	数据库列
person	Person/ProspectPerson.concernRoleID（使用 caseParticipantRoleID 进行计算）
gender	Person/ProspectPerson.gender
comments	不适用

4.7 标识

表 7. 标识映射

动态证据属性	数据库列
participant	ConcernRoleAlternateID.concernRoleID（使用 caseParticipantRoleID 进行计算）
alternateID	ConcernRoleAlternateID.alternateID
altIDType	ConcernRoleAlternateID.typeCode
fromDate	ConcernRoleAlternateID.startDate
toDate	ConcernRoleAlternateID.endDate
comments	ConcernRoleAlternateID.comments

4.8 名称

表 8. 名称映射

动态证据属性	数据库列
participant	AlternateName.concernRoleID（使用 caseParticipantRoleID 进行计算）
title	AlternateName.title
firstName	AlternateName.firstForename
middleName	AlternateName.otherForename
lastName	AlternateName.surname
suffix	AlternateName.nameSuffix
initials	AlternateName.initials
nameType	AlternateName.nameType
comments	AlternateName.comments

4.9 电话号码

表 9. 电话号码映射

动态证据属性	数据库列
participant	ConcernRolePhoneNumber.concernRoleID（使用 caseParticipantRoleID 进行计算）
phoneCountryCode	PhoneNumber.phoneCountryCode
phoneAreaCode	PhoneNumber.phoneAreaCode
phoneNumber	PhoneNumber.phoneNumber
phoneExtension	PhoneNumber.phoneExtension
fromDate	ConcernRolePhoneNumber.startDate
toDate	ConcernRolePhoneNumber.endDate
phoneType	ConcernRolePhoneNumber.typeCode
comments	PhoneNumber.comments

4.10 关系

表 10. 关系映射

动态证据属性	数据库列
participant	ConcernRoleRelationship.concernRoleID（使用 caseParticipantRoleID 进行计算）
relatedParticipant	ConcernRoleRelationship.relConcernRoleID
fromDate	ConcernRoleRelationship.startDate
toDate	ConcernRoleRelationship.endDate
relationshipType	ConcernRoleRelationship.relationshipType
endReason	ConcernRoleRelationship.relEndReasonCode
comments	ConcernRoleRelationship.comments

第 5 章 定制人员/潜在人员证据

5.1 简介

本章描述人员/潜在人员证据的可用定制选项和扩展点。这些选项和扩展点中的一部分或全部可能适用于您，取决于您的现有定制和配置。有五个需要考虑的主要方面，如下所示:

- 复制器
- 转换器
- 主要信息选择
- 相反证据
- 参与者数据案例负责人

对这些方面进行了详细说明并提供了示例。**请注意，这些仅为示例。**

5.2 复制器

5.2.1 什么是复制器?

复制器为兼顾向后兼容性而将证据上的更改反映到相关旧表中。复制器提取动态证据详细信息并将其转换为包含要存储在旧表中的详细信息的结构。随后会将这些详细信息写入相关数据库表中，从而确保旧表中的信息与主要数据源动态证据同步。为每种人员/潜在人员证据类型都提供了缺省复制器实现。这些缺省实现包含允许复制以下部分所含定制字段的扩展点。

注: 只会使用继承集中的最后一个版本来将数据复制到旧表。

5.2.2 为什么要扩展复制器?

在扩展了旧数据库表的情况下，可能需要扩展复制器。

5.2.3 复制器扩展

可以扩展随应用程序提供的复制器，以允许将人员/潜在人员证据复制到定制数据库列。接口可供所有提供的证据类型使用并且可在 curam.pdc.impl 包中找到，如下所列。可以根据证据类型撰写使用这些接口的定制实现。

复制器扩展器接口

- PDCAddressReplicatorExtender
- PDCAlternateIDReplicatorExtender
- PDCAlternateNameReplicatorExtender
- PDCBankAccountReplicatorExtender
- PDCBirthAndDeathReplicatorExtender
- PDCContactPreferencesReplicatorExtender
- PDCEmailAddressReplicatorExtender
- PDCGenderReplicatorExtender
- PDCPhoneNumberReplicatorExtender

- PDCRelationshipsReplicatorExtender

这些接口中的大多数具有一种方法 assignDynamicEvidenceToExtendedDetails。它接受以下两种参数:

- dynamicEvidenceDataDetails - 动态证据详细信息
- details - 包含旧表扩展详细信息的结构

PDCBirthAndDeathReplicatorExtender 和 PDCGenderReplicatorExtender 具有两种方法，assignDynamicEvidenceToExtendedPersonDetails 和 assignDynamicEvidenceToExtendedProspectPersonDetails。assignDynamicEvidenceToExtendedPersonDetails 接受以下两种参数:

- dynamicEvidenceDataDetails - 动态证据详细信息
- details - 包含旧表扩展人员详细信息的结构

assignDynamicEvidenceToExtendedProspectPersonDetails 也接受以下两种参数:

- dynamicEvidenceDataDetails - 动态证据详细信息
- details - 包含旧表扩展潜在人员详细信息的结构

5.2.4 示例: 实现人员/潜在人员证据复制器扩展器

下例概述了如何扩展复制器以将人员/潜在人员证据映射到定制字段。此例提供了 PDCPhoneNumberReplicatorExtender 扩展非常基本的实现。在此场景中，PhoneNumber 表已扩展并且包含定制字段“phoneProvider”。电话号码的动态证据配置也包含“phoneProvider”属性。此例假定 ParticipantPhoneDetails 结构已扩展为包含定制字段。有关动态证据配置的更多信息，请参阅 Cúram Dynamic Evidence Configuration Guide。定制复制器扩展实现负责将动态证据数据映射到已扩展 PhoneNumber 表上代表“phoneProvider”属性的结构属性。

注: 数据映射是必需的，缺省实现将会执行实际的数据复制。

扩展复制器涉及的步骤包括:

- 提供将定制数据映射回旧表的复制器扩展实现
- 为新的复制器扩展实现添加绑定

5.2.4.1 步骤 1: 提供复制器扩展实现

第一步是为证据类型提供用于实现相关复制扩展接口的新实现，将定制数据映射回旧表。下面的代码段演示了 PDCPhoneNumberReplicatorExtender 的定制实现。它只是将动态证据属性的值分配给 phoneProvider 结构属性。随后会使用 PDCPhoneNumberReplicatorExtender 的缺省实现将此信息与其他动态证据属性一起插入。

```
public class SampleReplicatorExtenderImpl
  implements PDCPhoneNumberReplicatorExtender {

  public void assignDynamicEvidenceToExtendedDetails(
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails,
    ParticipantPhoneDetails details)
    throws AppException, InformationalException {

    details.phoneProvider =
    dynamicEvidenceDataDetails.getAttribute("phoneProvider").getValue();
  }
}
```

5.2.4.2 步骤 2: 为新的复制器扩展实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Register the replicator extension implementation
  Multibinder<PDCPhoneNumberReplicatorExtender> sampleReplicatorExtender =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberReplicatorExtender.class);

  sampleReplicatorExtender.addBinding().to(SampleReplicatorExtenderImpl.class);
 }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.2.5 为什么要实现复制器?

在引入了新的动态证据类型的情况下，可能需要实现新复制器。

5.2.6 实现复制器

可以轻松地开发复制器以满足诸如新建动态证据类型等场景。下节提供了具体的示例，概述了要使新的复制器启动并运行所必需的步骤和工件。复制器实现通过基于事件的机制进行调用。在插入、修改或除去操作后激活动态证据时将抛出事件。对于新的证据类型，需要开发事件侦听器来侦听此事件，调用复制进程，本章的后续部分将对此进行更详细的讨论。下节演示了如何实现复制器。

5.2.7 示例: 实现人员/潜在人员证据复制器

下例概述了如何实现复制器。在此场景中，将“样本外国居留权”配置为新的动态证据类型。有关如何配置新证据类型的更多信息，请参阅 Cúram Dynamic Evidence Configuration Guide。新的“样本外国居留权”证据类型具有如下属性:

- participant - 证据要输入到的目标人员/潜在人员的案例参与者角色标识
- country - 居留权所属国家或地区
- fromDate - 居留权的起始日期
- toDate - 居留权的结束日期
- reason - 在此国家或地区的居留原因

假定此动态证据类型已激活并且已配置为供人员/潜在人员使用。截至目前，“样本外国居留权”信息已存储为 SampleForeignResidency 数据库表中的静态证据。现在必须将此信息存储为动态证据。新的复制器可能需要将证据更改复制到旧数据库表中，以使此表与动态证据同步。

实现复制器涉及的步骤包括:

- 为动态证据类型提供复制器接口
- 提供将把动态证据复制到旧数据库表的复制器实现
- 实现将触发复制的事件侦听器
- 为新的事件侦听器实现添加绑定

5.2.7.1 步骤 1: 提供复制器接口

新的复制器接口应包含三种方法:

`replicateInsertEvidence`，将已激活的插入“样本外国居留权”证据复制到“样本外国居留权”旧数据库表中。它接受以下一种参数:

- evidenceDescriptorDtls，已激活的证据描述符详细信息

replicateModifyEvidence，将已激活的修改“样本外国居留权”证据复制到“样本外国居留权”旧数据库表中。它接受以下两种参数:

- evidenceDescriptorDtls，已激活的证据描述符详细信息
- previousActiveEvidDescriptorDtls，修改前处于活动状态的证据的证据描述符详细信息

replicateRemoveEvidence，将已激活的除去“样本外国居留权”证据复制到“样本外国居留权”旧数据库表中。它接受以下一种参数:

- evidenceDescriptorDtls，已激活的证据描述符详细信息

```
@ImplementedBy(SampleForeignResidencyReplicatorImpl.class)
public interface SampleForeignResidencyReplicator {

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateModifyEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls,
    final EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateRemoveEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;
}
```

5.2.7.2 步骤 2: 提供复制器实现

复制器实现应为上一节中描述的三种方法提供实现。这些方法应将动态证据数据转换为适于写入旧数据库表的数据，并为此证据类型更新旧数据库表。

```
public class SampleForeignResidencyReplicatorImpl
  implements SampleForeignResidencyReplicator {

  protected SampleForeignResidencyReplicatorImpl() {
  }

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();
    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();
    UniqueID uniqueIDObj = UniqueIDFactory.newInstance();

    EvidenceControllerInterface evidenceControllerObj =
      (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();
```

```
  sampleForeignResidencyDtls.startDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("fromDate"));

  sampleForeignResidencyDtls.endDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("toDate"));

  sampleForeignResidencyDtls.reasonCode =
    dynamicEvidenceDataDetails.getAttribute("reason").getValue();

  sampleForeignResidencyDtls.concernRoleID = evidenceDescriptorDtls.participantID;
  sampleForeignResidencyDtls.foreignResidencyID = uniqueIDObj.getNextID();
  sampleForeignResidencyDtls.statusCode = RECORDSTATUS.NORMAL;

  sampleForeignResidencyObj.insert(sampleForeignResidencyDtls);
}

public void replicateModifyEvidence(
  final EvidenceDescriptorDtls evidenceDescriptorDtls,
  final EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
  throws AppException, InformationalException {

  List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
    new ArrayList<SampleForeignResidencyKey>();

  SampleForeignResidencyDtls sampleForeignResidencyDtls =
 new SampleForeignResidencyDtls();

  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

  EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
  eiEvidenceKey.evidenceID = previousActiveEvidDescriptorDtls.relatedID;
  eiEvidenceKey.evidenceType = previousActiveEvidDescriptorDtls.evidenceType;

  EIEvidenceReadDtls eiEvidenceReadDtls =
    evidenceControllerObj.readEvidence(eiEvidenceKey);

  DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
    (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

  sampleForeignResidencyDtls.countryCode =
    dynamicEvidenceDataDetails.getAttribute("country").getValue();

  sampleForeignResidencyDtls.startDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("fromDate"));

  sampleForeignResidencyDtls.endDate =
 (Date) DynamicEvidenceTypeConverter.convert(
    dynamicEvidenceDataDetails.getAttribute("toDate"));

  sampleForeignResidencyDtls.reasonCode =
 dynamicEvidenceDataDetails.getAttribute("reason").getValue();

  SampleForeignResidency sampleForeignResidencyObj =
 SampleForeignResidencyFactory.newInstance();

  SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
    new SampleForeignResidencyReadMultiKey();
  sampleForeignResidencyReadMultiKey.concernRoleID =
 previousActiveEvidDescriptorDtls.participantID;

  SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyReadMultiDtlsList =
    sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);
```

```
  for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls :
 sampleForeignResidencyReadMultiDtlsList.dtls) {

    if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
   sampleForeignResidencyDtls.countryCode))
      && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
sampleForeignResidencyDtls.reasonCode))) {

      SampleForeignResidencyKey sampleForeignResidencyKey = new SampleForeignResidencyKey();
      sampleForeignResidencyKey.sampleForeignResidencyID =
  sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

      sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
    }
  }

  for (SampleForeignResidencyKey sampleForeignResidencyKey : sampleForeignResidencyKeyList) {

    sampleForeignResidencyDtls = new SampleForeignResidencyDtls();

    eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

    eiEvidenceReadDtls = evidenceControllerObj.readEvidence(eiEvidenceKey);

    dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate = (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate = (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
      dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    sampleForeignResidencyDtls.concernRoleID = evidenceDescriptorDtls.participantID;

    SampleForeignResidencyDtls sampleForeignResidencyReadDtls =
   sampleForeignResidencyObj.read(sampleForeignResidencyKey);

    sampleForeignResidencyReadDtls.assign(sampleForeignResidencyDtls);

    sampleForeignResidencyObj.modify(sampleForeignResidencyKey, sampleForeignResidencyReadDtls);
  }
}

public void replicateRemoveEvidence(
  final EvidenceDescriptorDtls evidenceDescriptorDtls)
  throws AppException, InformationalException {

  List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
 new ArrayList<SampleForeignResidencyKey>();

  SampleForeignResidencyDtls sampleForeignResidencyDtls =
 new SampleForeignResidencyDtls();

  EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

  EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
```

```
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
   dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID =
   evidenceDescriptorDtls.participantID;

    SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyReadMultiDtlsList =
      sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);

    for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls :
   sampleForeignResidencyReadMultiDtlsList.dtls) {

      if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
     sampleForeignResidencyDtls.countryCode))
        && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
  sampleForeignResidencyDtls.reasonCode))) {

        SampleForeignResidencyKey sampleForeignResidencyKey = new SampleForeignResidencyKey();
        sampleForeignResidencyKey.sampleForeignResidencyID =
    sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

        sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
      }
    }

    for (SampleForeignResidencyKey sampleForeignResidencyKey : sampleForeignResidencyKeyList) {

      sampleForeignResidencyDtls = sampleForeignResidencyObj.read(sampleForeignResidencyKey);
      sampleForeignResidencyDtls.statusCode = RECORDSTATUS.CANCELLED;
      sampleForeignResidencyObj.modify(sampleForeignResidencyKey, sampleForeignResidencyDtls);
    }
  }
}
```

5.2.7.3 步骤 3: 实现事件侦听器

需要实现新的事件侦听器，以侦听因激活证据而发生的“样本外国居留权”类型的事件。此侦听器应实现 curam.pdc.impl.PDCEvents 接口并为这三种方法提供实现。这就是复制过程以及任何其他可能需要的定制处理的起始点。

```
public class SampleForeignResidencyEventsListener
  implements PDCEvents {

  @Inject
  private SampleForeignResidencyReplicator sampleForeignResidencyReplicator;

  public void insertedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateInsertEvidence(evidenceDescriptorDtls);
    }
  }

  public void modifiedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls,
    EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateModifyEvidence(evidenceDescriptorDtls,
      previousActiveEvidDescriptorDtls);
    }
  }

  public void removedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

 if (evidenceDescriptorDtls.evidenceType.equals("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateRemoveEvidence(evidenceDescriptorDtls);
    }
  }
}
```

5.2.7.4 步骤 4: 为新事件侦听器实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the event listener
    Multibinder<PDCEvents> sampleEventListeners =
      Multibinder.newSetBinder(binder(), PDCEvents.class);

    sampleEventListeners.addBinding().to(
      SampleForeignResidencyEventsListener.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.3 转换器

5.3.1 什么是转换器?

转换器就是用于将旧的人员/潜在人员转换为动态证据的机制。如果已在申请外通过使用 Cúram Data Manager（DMX 文件）等工具对旧数据库表进行了填充，那么可以使用转换器来将此数据转换为动态证据。为每种人员/潜在人员证据类型都提供了缺省转换器实现。这些缺省实现包含允许转换以下部分所含定制字段的扩展点。

5.3.2 为什么要扩展转换器?

在扩展了旧数据库表的情况下，可能需要扩展转换器。转换器通常只用于开发环境或升级工具，并且不应作为日常处理的一部分。

5.3.3 转换器扩展

可对应用程序随附的转换器进行扩展，以允许将定制数据库列转换为人员/潜在人员动态证据。接口可供所有证据类型使用并且可在 curam.pdc.impl 包中找到，如下所列。可以根据证据类型撰写使用这些接口的定制实现。

转换器扩展（填充器）接口

- PDCAddressEvidencePopulator
- PDCAlternateIDEvidencePopulator
- PDCAlternateNameEvidencePopulator
- PDCBankAccountEvidencePopulator
- PDCBirthAndDeathEvidencePopulator
- PDCContactPreferencesEvidencePopulator
- PDCEmailAddressEvidencePopulator
- PDCGenderEvidencePopulator
- PDCPhoneNumberEvidencePopulator
- PDCRelationshipsEvidencePopulator

这些接口中的大多数具有一种方法 `populate`。根据证据类型的不同，它接受的参数也不同。

`PDCBirthAndDeathEvidencePopulator` 和 `PDCGenderEvidencePopulator` 接口具有两种方法，`populatePerson` 和 `populateProspectPerson`。

`populatePerson` 接受以下四种参数:

- concernRoleKey - 此证据的相关关注角色的唯一标识
- caseIDKey - 参与者数据案例的唯一标识
- personDtls - 包含旧表中扩展人员详细信息的结构
- dynamicEvidenceDataDetails - 动态证据详细信息

`populateProspectPerson` 也接受以下四种参数:

- concernRoleKey - 此证据的相关关注角色的唯一标识
- caseIDKey - 参与者数据案例的唯一标识
- prospectPersonDtls - 包含旧表中扩展潜在人员详细信息的结构
- dynamicEvidenceDataDetails - 动态证据详细信息

5.3.4 示例: 实现人员/潜在人员证据填充器

下例概述了如何扩展转换器以将定制数据库列映射到人员/潜在人员证据。此示例提供了 PDCPhoneNumberEvidencePopulator 扩展非常基本的实现。在此场景中，PhoneNumber 表已扩展并且包含定制列“phoneProvider”。电话号码的动态证据配置也包含“phoneProvider”属性。定制填充器实现负责将表示扩展 PhoneNumber 表中“phoneProvider”属性的结构属性转换为动态证据数据。有关动态证据配置的更多信息，请参阅 Cúram Dynamic Evidence Configuration Guide。

注: 数据转换是必需的，缺省转换器将会留意动态证据的实际存储。

扩展转换器涉及的步骤包括:

- 提供将把旧表中的定制字段转换为动态证据数据的填充器实现
- 为新的填充器实现添加绑定

5.3.4.1 步骤 1: 提供填充器实现

第一步是为证据类型提供用于实现相关填充器接口的新实现，将旧表中的定制字段转换为动态证据。下面的代码段演示 PDCPhoneNumberEvidencePopulator 的定制实现，它只是把 phoneProvider 结构属性转换为动态证据的等同属性。随后会使用缺省转换器实现将此动态证据与其他动态证据属性一起进行存储。

```
public class SamplePopulatorImpl
  implements PDCPhoneNumberEvidencePopulator {

  public void populate(
    ConcernRoleKey concernRoleKey, CaseIDKey caseIDKey,
    ConcernRolePhoneNumberDtls concernRolePhoneNumberDtls,
    PhoneNumberDtls phoneNumberDtls,
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails)
    throws AppException, InformationalException {

    DynamicEvidenceDataAttributeDetails phoneProvider =
      dynamicEvidenceDataDetails.getAttribute("phoneProvider");

    DynamicEvidenceTypeConverter.setAttribute(phoneProvider,
      phoneNumberDtls.phoneProvider);
  }
}
```

5.3.4.2 为新的填充器实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Register the populator implementation
  Multibinder<PDCPhoneNumberEvidencePopulator> samplePopulator =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberEvidencePopulator.class);

  samplePopulator.addBinding().to(SamplePopulatorImpl.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.3.5 为什么要实现转换器?

在引入了新的动态证据类型的情况下，可能需要实现新转换器。转换器通常只用于开发环境或升级工具，并且不应作为日常处理的一部分。

5.3.6 实现转换器

可使用 PDCConverter 接口来开发转换器实现。可在 curam.pdc.impl 包中找到 PDCConverter 接口。此接口具有一种方法 storeEvidence。它接受以下两种参数:

- concernRoleKey - 关注角色的唯一标识
- caseIDKey - 参与者数据案例的唯一标识。

下节演示了如何实现转换器。

5.3.7 示例: 实现人员/潜在人员证据转换器

下例概述了如何实现转换器。在此场景中，将“样本外国居留权”配置为新的动态证据类型。有关如何配置新证据类型的更多信息，请参阅 Cúram Dynamic Evidence Configuration Guide。新的“样本外国居留权”证据类型具有如下属性:

- participant - 证据要输入到的目标人员/潜在人员的案例参与者角色标识
- country - 居留权所属国家或地区
- fromDate - 居留权的起始日期
- toDate - 居留权的结束日期
- reason - 在此国家或地区的居留原因

假定此动态证据类型已激活并且已配置为供人员/潜在人员使用。“样本外国居留权”信息先前已存储为 SampleForeignResidency 数据库表中的静态证据。现在必须将此信息存储为动态证据。需要新的转换器来从旧表中提取此信息并将其转换并存储为动态证据。

实现转换器涉及的步骤包括:

- 提供将旧数据转换为动态证据的转换器实现
- 为新的转换器实现添加绑定

5.3.7.1 步骤 1: 提供转换器实现

下面的代码段演示了 PDCConverter 的实现。它从旧的 SampleForeignResidency 表中检索出某个人员/潜在人员的所有“样本外国居留权”信息，将此信息转换为动态证据数据结构并存储产生的信息。

```
public class SampleForeignResidencyConverterImpl
  implements PDCConverter {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public void storeEvidence(ConcernRoleKey concernRoleKey, CaseIDKey caseIDKey)
    throws AppException, InformationalException {

    PDCCaseIDCaseParticipantRoleID pdcCaseIDCaseParticipantRoleID =
      new PDCCaseIDCaseParticipantRoleID();

    ParticipantDataCase participantDataCaseObj = ParticipantDataCaseFactory.newInstance();
    pdcCaseIDCaseParticipantRoleID.caseID =
      participantDataCaseObj.getParticipantDataCase(concernRoleKey).caseID;

    CaseIDTypeCodeKey caseIDTypeCodeKey = new CaseIDTypeCodeKey();
    caseIDTypeCodeKey.caseID = pdcCaseIDCaseParticipantRoleID.caseID;
    caseIDTypeCodeKey.typeCode = CASEPARTICIPANTROLETYPE.PRIMARY;

    pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID =
```

```
      CaseParticipantRoleFactory.newInstance().readByCaseIDAndTypeCode(caseIDTypeCodeKey).
dtls.caseParticipantRoleID;

    SampleForeignResidency sampleForeignResidencyObj = SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID = concernRoleKey.concernRoleID;

    SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyList =
      sampleForeignResidencyObj.searchByConcernRole(sampleForeignResidencyReadMultiKey);

   for (SampleForeignResidencyReadMultiDtls sampleForeignResidencyReadMultiDtls :
sampleForeignResidencyList.dtls) {

      final EvidenceTypeKey eType = new EvidenceTypeKey();
      eType.evidenceType = "SampleForeignResidency";

      EvidenceTypeDef evidenceType =
        etDefDAO.readActiveEvidenceTypeDefByTypeCode(eType.evidenceType);

      EvidenceTypeVersionDef evTypeVersion =
        etVerDefDAO.getActiveEvidenceTypeVersionAtDate(evidenceType, Date.getCurrentDate());

      DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
        DynamicEvidenceDataDetailsFactory.newInstance(evTypeVersion);

      DynamicEvidenceDataAttributeDetails participant =
        dynamicEvidenceDataDetails.getAttribute("participant");

      DynamicEvidenceTypeConverter.setAttribute(participant,
        pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID);

      DynamicEvidenceDataAttributeDetails country =
        dynamicEvidenceDataDetails.getAttribute("country");

      DynamicEvidenceTypeConverter.setAttribute(country,
        sampleForeignResidencyReadMultiDtls.countryCode);

      DynamicEvidenceDataAttributeDetails fromDate =
        dynamicEvidenceDataDetails.getAttribute("fromDate");

      DynamicEvidenceTypeConverter.setAttribute(fromDate, sampleForeignResidencyReadMultiDtls.startDate);

      DynamicEvidenceDataAttributeDetails endDate =
        dynamicEvidenceDataDetails.getAttribute("toDate");

      DynamicEvidenceTypeConverter.setAttribute(endDate, sampleForeignResidencyReadMultiDtls.endDate);

      DynamicEvidenceDataAttributeDetails reasonCode =
        dynamicEvidenceDataDetails.getAttribute("reason");

      DynamicEvidenceTypeConverter.setAttribute(reasonCode, sampleForeignResidencyReadMultiDtls.reasonCode);

      EvidenceControllerInterface evidenceControllerObj =
        (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

      EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls = new EvidenceDescriptorInsertDtls();
      evidenceDescriptorInsertDtls.participantID = concernRoleKey.concernRoleID;
      evidenceDescriptorInsertDtls.evidenceType = eType.evidenceType;
      evidenceDescriptorInsertDtls.receivedDate = curam.util.type.Date.getCurrentDate();
      evidenceDescriptorInsertDtls.caseID = pdcCaseIDCaseParticipantRoleID.caseID;

      EIEvidenceInsertDtls eiEvidenceInsertDtls = new EIEvidenceInsertDtls();

      eiEvidenceInsertDtls.descriptor.assign(evidenceDescriptorInsertDtls);
      eiEvidenceInsertDtls.descriptor.participantID = concernRoleKey.concernRoleID;
```

```
      eiEvidenceInsertDtls.descriptor.changeReason = EVIDENCECHANGEREASON.REPORTEDBYCLIENT;
      eiEvidenceInsertDtls.evidenceObject = dynamicEvidenceDataDetails;

      evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);
    }
  }
}
```

5.3.7.2 步骤 2: 为新的转换器实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

  public void configure() {

  // Register the converter implementation
  Multibinder<PDCConverter> sampleForeignResidencyConverter =
    Multibinder.newSetBinder(binder(), PDCConverter.class);

  sampleForeignResidencyConverter.addBinding().to(SampleForeignResidencyConverterImpl.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.4 主要信息选择

旧参与者管理器实体具有主要指示符的概念，在创建证据时用户可以指定表示主要数据的银行帐户/电话号码等。这不适用于动态证据类型。不再是由用户指定主要记录；而是由后台中的算法来计算哪条记录应为主要记录。这些算法基于已定义的业务策略并且可进行修改，下节概述了这些算法的详细信息。

5.4.1 为什么要更改主要信息选择?

由于主要记录的标识不再由用户决定，所以如果当缺省业务策略不可取时，就可能需要修改此选择过程。

5.4.2 更改主要信息选择

确定哪些数据应选为主要信息的策略，可以通过使用缺省主要处理程序接口来修改。为每个在其旧表上具有主要标识的已提供动态证据类型都定义了接口，这些接口可在 curam.pdc.impl 包中找到，列示如下。

主要处理程序实现通过基于事件的机制进行调用。在插入、修改或除去操作后激活动态证据时将抛出事件。对于新的证据类型，需要开发事件侦听器来侦听此事件，调用确定主要数据的适当算法，本章的后续部分将对此进行更详细的讨论。下节演示了如何实现主要处理程序。

主要处理程序接口

- PDCPrimaryAddressHandler
- PDCPrimaryAlternateIDHandler
- PDCPrimaryAlternateNameHandler
- PDCPrimaryBankAccountHandler
- PDCPrimaryEmailAddressHandler
- PDCPrimaryPhoneNumberHandler

5.4.3 更改主要信息选择示例

下例概述了如何实现主要处理程序。在此场景中，为选择主要电话号码而定义的业务策略会选择开始日期最近的电话号码。

实现主要处理程序涉及的步骤包括:

- 提供将识别主要记录的主要处理程序实现
- 为新的主要处理程序实现添加绑定

5.4.3.1 步骤 1: 提供主要处理程序实现

第一步是为证据类型提供用于实现相关主要处理程序接口的新实现，以及标识主要记录。下面的代码段演示了 PDCPrimaryPhoneNumberHandler 的实现，它只是提取开始日期最近的电话号码并将其设置为主要记录。

```
public class SamplePrimaryPhoneNumberHandlerImpl
  implements PDCPrimaryPhoneNumberHandler {

  protected SamplePrimaryPhoneNumberHandlerImpl() {
  }

  public void setPrimaryPhoneNumber(EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    ConcernRoleKey concernRoleKey = new ConcernRoleKey();
    concernRoleKey.concernRoleID = evidenceDescriptorDtls.participantID;

    ConcernRolePhoneNumberDtlsList concernRolePhoneNumberDtlsList =
      ConcernRolePhoneNumberFactory.newInstance().searchByConcernRole(concernRoleKey);

    ConcernRole concernRoleObj = ConcernRoleFactory.newInstance();
    ConcernRoleDtls concernRoleDtls = concernRoleObj.read(concernRoleKey);
    Date currentPrimaryPhoneNumberStartDate = Date.kZeroDate;

    List<SampleSortedPhoneNumber> list = new ArrayList<SampleSortedPhoneNumber>();

    for (ConcernRolePhoneNumberDtls concernRolePhoneNumberDtls:concernRolePhoneNumberDtlsList.dtls) {

      PhoneNumberKey phoneNumberKey = new PhoneNumberKey();
      phoneNumberKey.phoneNumberID = concernRolePhoneNumberDtls.phoneNumberID;

      if (concernRolePhoneNumberDtls.phoneNumberID == concernRoleDtls.primaryPhoneNumberID) {
        currentPrimaryPhoneNumberStartDate = concernRolePhoneNumberDtls.startDate;
      }

      SampleSortedPhoneNumber sampleSortedPhoneNumber =
        new SampleSortedPhoneNumber(concernRolePhoneNumberDtls);
      list.add(sampleSortedPhoneNumber);
    }

    Collections.sort(list);

    SampleSortedPhoneNumber newPrimaryPhoneNumber = list.get(0);

    if (newPrimaryPhoneNumber.getStartDate().after(currentPrimaryPhoneNumberStartDate)) {
      concernRoleDtls.primaryPhoneNumberID = newPrimaryPhoneNumber.getPhoneNumberID();
      concernRoleObj.pdcModify(concernRoleKey, concernRoleDtls);
    }
}
  class SampleSortedPhoneNumber implements Comparable<SampleSortedPhoneNumber> {
    private long phoneNumberID;
    private Date startDate;

    SampleSortedPhoneNumber(ConcernRolePhoneNumberDtls dtls) {
      this.phoneNumberID = dtls.phoneNumberID;
      this.startDate = dtls.startDate;
```

```
    }

    public long getPhoneNumberID() {
      return phoneNumberID;
    }

    public Date getStartDate() {
      return startDate;
    }

    public int compareTo(SampleSortedPhoneNumber o) {
      return o.getStartDate().compareTo(this.getStartDate());
    }
  }
}
```

5.4.3.2 步骤 2: 为新的主要处理程序实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the primary handler implementation
    bind(PDCPrimaryPhoneNumberHandler.class).to(
      SamplePrimaryPhoneNumberHandlerImpl.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.5 相反证据

5.5.1 什么是相反证据?

相反证据是指一种证据类型，它包含必须一起处理的两个证据。关系动态证据类型就是相反证据的一个示例。将人员 A 记录为人员 B 的配偶时，与之对应的关系证据会将人员 B 记录为人员 A 的配偶。在为人员 A 插入、修改或除去证据时，系统会为人员 B 插入、修改或除去对应的关系证据。

5.5.2 为什么要提供相反证据实现?

如果开发了新的相反动态证据类型，那么还必须提供 ReciprocalEvidenceConversion 接口的实现。

5.5.3 相反证据实现

当插入、修改或除去证据时，将调用挂钩点，缺省情况下会触发相反证据处理程序功能。这个新的证据挂钩点称为 GlobalEvidenceHook，可在 curam.core.sl.infrastructure.impl 包中找到。GlobalEvidenceHook 接口允许在证据操作完成后进行定制操作。

GlobalEvidenceHook 接口

GlobalEvidenceHook 接口包含以下方法:

postInsertEvidence，在证据插入后调用，接受以下两种参数:

- caseKey - 证据所属案例的标识
- evKey - 证据的标识和类型

postModifyEvidence，在证据修改后调用，接受以下两种参数:

- caseKey - 证据所属案例的标识
- evKey - 证据的标识和类型

postRemoveEvidence，在证据除去后调用，接受以下两种参数:

- caseKey - 证据所属案例的标识
- evKey - 证据的标识和类型

postDiscardPendingUpdate，在放弃证据暂挂更新后调用，接受以下两种参数:

- caseKey - 证据所属案例的标识
- evKey - 证据的标识和类型

postDiscardPendingRemove，在放弃证据暂挂除去后调用，接受以下两种参数:

- caseKey - 证据所属案例的标识
- evKey - 证据的标识和类型

相反证据处理程序

GlobalEvidenceHook 的缺省实现会调用相反证据处理程序功能。相反证据处理程序负责处理所有公共相反证据。它会查找相反证据，然后在找到后对相反证据执行与原始证据相同的更改。如果找不到相反证据，并且已插入原始证据，那么它将插入相应的相反证据。由于相反证据处理程序是相反证据处理的核心，所以当前不可对其进行直接定制，但如有必要可通过 GlobalEvidenceHook 方式进行定制。

相反证据转换接口

ReciprocalEvidenceConversion 接口负责相反证据和原始证据的比较、参与者检索以及根据原始证据创建新的或修改后的相反证据。为制定定制相反证据，必须提供 ReciprocalEvidenceConversion 接口实现。如果处理程序不了解内部证据结构和转换接口实现，那么此时就需要主定制点。可在 curam.core.sl.infrastructure.impl 包中找到 ReciprocalEvidenceConversion 接口，此接口包含以下方法:

- Object getReciprocal(final Object original, final long targetCaseID) - 根据原始证据详细信息来创建相反证据详细信息
- Object getUpdatedReciprocal(final Object original, final Object unmodifiedReciprocal) - 根据原始证据详细信息和更改前的相反证据详细信息来创建更改后的相反证据详细信息
- long getPrimaryParticipant(final Object originalEvidence) - 检索原始证据中的主要参与者（关注角色标识）。请注意，原始证据上的主要参与者是相反证据上的相关参与者
- long getRelatedParticipant(final Object originalEvidence) - 检索原始证据中的相关参与者（关注角色标识）。请注意，原始证据上的相关参与者是相反证据上的主要参与者
- boolean matchEvidenceDetails(final Object evidenceDetails1, final Object evidenceDetails2) - 检查证据详细信息的匹配。实现此方法将确定传递的两个证据详细信息是否将视为相互匹配。

下节演示了如何实现相反证据。

5.5.4 相反证据实现示例

下例演示了相反证据实现。在此场景中，工作关系已配置为新的动态证据类型。有关如何配置新证据类型的更多信息，请参阅 Cúram Dynamic Evidence Configuration Guide。新的工作关系证据类型已标识为相反证据，并且具有下列属性:

- participant - 证据要输入到的目标人员/潜在人员的案例参与者角色标识

- relatedParticipant - 相关人员/潜在人员的案例参与者角色标识
- workingRelationship - 两个参与者之间的工作关系

假定此动态证据类型已激活并且已配置为供人员/潜在人员使用。为使此相反证据能由基础结构正确进行处理，必须提供 ReciprocalEvidenceConversion 接口的实现。

涉及的步骤包括:

- 提供相反证据转换实现
- 为新的相反证据转换实现添加绑定

5.5.4.1 步骤 1: 提供相反证据转换实现

```
public class SampleWorkingRelationshipReciprocalConversion
  implements ReciprocalEvidenceConversion {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public SampleWorkingRelationshipReciprocalConversion() {

  }

  public Object getReciprocal(
    final Object original, final long targetCaseID)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails = (DynamicEvidenceDataDetails) original;

    String workingRelationshipOriginal =
      originalDetails.getAttribute("workingRelationship").getValue();

    String workingRelationshipRec = "";

    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    EvidenceTypeKey evdType = new EvidenceTypeKey();
    evdType.evidenceType = "WORKINGRELATIONSHIP";

    EvidenceTypeDef evdTypeDef =
      etDefDAO.readActiveEvidenceTypeDefByTypeCode(evdType.evidenceType);

    EvidenceTypeVersionDef evTypeVersion =
      etVerDefDAO.getActiveEvidenceTypeVersionAtDate(evdTypeDef, Date.getCurrentDate());

    DynamicEvidenceDataDetails reciprocalDetails =
      DynamicEvidenceDataDetailsFactory.newInstance(evTypeVersion);

    DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
      reciprocalDetails.getAttribute("workingRelationship");

    DynamicEvidenceTypeConverter.setAttribute(workingRelationshipAttr, workingRelationshipRec);

    DynamicEvidenceDataAttributeDetails participantAttr =
      reciprocalDetails.getAttribute("participant");

    DynamicEvidenceTypeConverter.setAttribute(participantAttr,
        originalDetails.getAttribute("relatedParticipant").getValue());

    DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
```

```
      reciprocalDetails.getAttribute("relatedParticipant");

    DynamicEvidenceTypeConverter.setAttribute(relatedParticipantAttr,
        originalDetails.getAttribute("participant").getValue());

    return reciprocalDetails;
  }

  public Object getUpdatedReciprocal(
    final Object original, final Object unmodifiedReciprocal)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) original;
    DynamicEvidenceDataDetails reciprocalDetails =
      (DynamicEvidenceDataDetails) unmodifiedReciprocal;

    long caseParticipantRoleIDRec = Long.parseLong(
      reciprocalDetails.getAttribute("participant").getValue());
    long relCaseParticipantRoleIDRec = Long.parseLong(
      reciprocalDetails.getAttribute("relatedParticipant").getValue());
    String workingRelationshipRec = reciprocalDetails.getAttribute("workingRelationship").getValue();

    for (final DynamicEvidenceDataAttributeDetails listDetails: originalDetails.getAttributes()) {
      reciprocalDetails.getAttribute(listDetails.getName()).setValue(
        listDetails.getValue());
    }

    DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
      reciprocalDetails.getAttribute("workingRelationship");

    DynamicEvidenceTypeConverter.setAttribute(workingRelationshipAttr, workingRelationshipRec);

    DynamicEvidenceDataAttributeDetails participantAttr =
      reciprocalDetails.getAttribute("participant");

    DynamicEvidenceTypeConverter.setAttribute(participantAttr,
      caseParticipantRoleIDRec);

    DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
      reciprocalDetails.getAttribute("relatedParticipant");

    DynamicEvidenceTypeConverter.setAttribute(relatedParticipantAttr,
      relCaseParticipantRoleIDRec);

    return reciprocalDetails;
  }

  public boolean matchEvidenceDetails(
    final Object evidenceDetails1, final Object evidenceDetails2)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails1 =
      (DynamicEvidenceDataDetails) evidenceDetails1;
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails2 =
      (DynamicEvidenceDataDetails) evidenceDetails2;

    curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
      curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();

    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails1.getAttribute("participant"));

    Long concernRoleID1 =
      caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey)
.participantRoleID;
```

```
    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails1.getAttribute("relatedParticipant"));

    Long relConcernRoleID1 =
      caseParticipantRoleObj.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails2.getAttribute("participant"));

    Long concernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails2.getAttribute("relatedParticipant"));

    Long relConcernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    return dynamicEvidenceDataDetails1.getAttribute("workingRelationship").getValue().equals(
      dynamicEvidenceDataDetails2.getAttribute("workingRelationship").getValue())
      && (concernRoleID1.longValue() == concernRoleID2.longValue())
      && (relConcernRoleID1.longValue() == relConcernRoleID2.longValue());
  }

  public boolean matchOriginalAndReciprocal(
    final Object originalEvidence, final Object reciprocalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) originalEvidence;

    DynamicEvidenceDataDetails reciprocalDetails =
      (DynamicEvidenceDataDetails) reciprocalEvidence;

    curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
      curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();
    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      originalDetails.getAttribute("participant"));

    Long concernRoleID1 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      originalDetails.getAttribute("relatedParticipant"));

    Long relConcernRoleID1 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      reciprocalDetails.getAttribute("participant"));

    Long concernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID = (Long) DynamicEvidenceTypeConverter.convert(
      reciprocalDetails.getAttribute("relatedParticipant"));

    Long relConcernRoleID2 = caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    String workingRelationshipOriginal =
      originalDetails.getAttribute("workingRelationship").getValue();

    String workingRelationshipRec = "";
```

```
    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    return reciprocalDetails.getAttribute("workingRelationship").getValue().equals(
      workingRelationshipRec)
      && (concernRoleID1.longValue() == relConcernRoleID2.longValue())
      && (relConcernRoleID1.longValue() == concernRoleID2.longValue());
  }

  public long getPrimaryParticipant(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails = (DynamicEvidenceDataDetails)
originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("participant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID = caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance().read
(caseParticipantRoleKey).participantRoleID;
  }

  public long getRelatedParticipant(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("relatedParticipant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey = new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID = caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance().read
(caseParticipantRoleKey).participantRoleID;
  }
}
```

5.5.4.2 步骤 2: 为新的相反证据转换实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

  public void configure() {

    MapBinder<CASEEVIDENCEEntry, ReciprocalEvidenceConversion> recEvidenceConversionMapBinder =
      MapBinder.newMapBinder(binder(), CASEEVIDENCEEntry.class, ReciprocalEvidenceConversion.class);

    reciprocalEvidenceConversionMapBinder.addBinding(
   CASEEVIDENCEEntry.get("WORKINGRELATIONSHIP")).to(
      SampleWorkingRelationshipReciprocalConversion.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

5.5.5 相反证据限制

相反证据处理基础结构具有以下限制:

- 证据必须是临时证据。可以是静态证据、动态证据或已生成的证据.
- 证据必须具有参与者和相关参与者，或者 ReciprocalEvidenceConversion 实现代码必须能够使用证据详细信息确定参与者和相关参与者。
- 如果相反证据及其关联原始证据位于同一案例上，那么它们的更改必须始终一起应用，否则原始证据和相反证据数据将不同步。
- 仅当相关参与者都在同一案例上注册为 MEMBER 或 PRIMARY 参与者或者将证据记录为人员/潜在人员证据时，才能自动处理相反证据。

5.6 参与者数据案例负责人

5.6.1 为什么要更改参与者数据案例负责人?

如果需要对参与者的所有权实施更严格的控制，那么可能需要更改参与者数据案例负责人。

5.6.2 更改参与者数据案例负责人

如果已在系统上注册人员/潜在人员，那么会在后台创建一个案例以帮助管理此数据，此案例也称为“参与者数据案例”。缺省情况下，此案例的负责人是当前的登录用户。可以通过 PDCCaseOwnerAssignmentStrategy 接口来将此负责人更改为其他案例负责人。可在 curam.pdc.impl 包中找到 PDCCaseOwnerAssignmentStrategy 接口，此接口具有一种方法 createOwner。它接受以下两种参数:

- key - 参与者数据案例的标识
- ownerDtls - 参与者数据案例负责人的详细信息

5.6.3 更改参与者数据案例负责人示例

下例概述如何更改参与者数据案例负责人，在此场景中负责人设置为系统用户。

涉及的步骤包括:

- 提供将设置案例负责人的案例负责人分配策略实现
- 为案例负责人分配策略实现添加绑定

5.6.3.1 步骤 1: 提供案例负责人分配策略实现

下面的代码段演示 PDCCaseOwnerAssignmentStrategy 的样本实现，它只是将负责人设置为系统用户。

```
@Singleton
public class SampleCaseOwnerAssignmentStrategyImpl
  implements PDCCaseOwnerAssignmentStrategy {

  public void createOwner(CaseHeaderKey key, OrgObjectLinkDtls ownerDtls)
    throws AppException, InformationalException {

    ownerDtls.orgObjectType = ORGOBJECTTYPE.USER;
    ownerDtls.userName = UserAccessFactory.newInstance().getSystemUserDetails().userName;

    OrgObjectLinkFactory.newInstance().insert(ownerDtls);

    OrgObjectLinkKey orgObjectLinkKey = new OrgObjectLinkKey();
    orgObjectLinkKey.orgObjectLinkID = ownerDtls.orgObjectLinkID;

    CaseUserRoleDtls caseUserRoleDtls = new CaseUserRoleDtls();
    caseUserRoleDtls.caseID = key.caseID;
```

```
    caseUserRoleDtls.orgObjectLinkID = orgObjectLinkKey.orgObjectLinkID;
    caseUserRoleDtls.typeCode = CASEUSERROLETYPE.OWNER;
    caseUserRoleDtls.recordStatus = RECORDSTATUS.NORMAL;

    curam.core.sl.entity.fact.CaseUserRoleFactory.newInstance().insert(caseUserRoleDtls);

    CaseHeader caseHeaderObj = CaseHeaderFactory.newInstance();
    CaseHeaderDtls caseHeaderDtls = caseHeaderObj.read(key);
    caseHeaderDtls.ownerOrgObjectLinkID = orgObjectLinkKey.orgObjectLinkID;
    caseHeaderObj.modify(key, caseHeaderDtls);
  }
}
```

5.6.3.2 步骤 2: 为案例负责人分配策略实现添加绑定

使用 Guice 绑定注册实现。

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the implementation
    bind(PDCCaseOwnerAssignmentStrategy.class)
   .to(SampleCaseOwnerAssignmentStrategyImpl.class);
  }
}
```

注: 必须通过向 ModuleClassName 数据库表添加行来注册新的 Guice 模块。请参阅 Persistence Cookbook 以获取更多信息。

声明

本信息是为在美国提供的产品和服务编写的。IBM 可能在其它国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务的操作，由用户自行负责。IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并不意味着授予用户使用这些专利的任何许可。您可以用书面形式将许可查询寄往:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

有关双字节 (DBCS) 信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

本条款不适用英国或这样的条款与当地法律不一致的任何国家或地区: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销或适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息可能包含技术方面不够准确的地方或印刷错误。本信息将定期更改；这些更改将编入本信息的新版本中。IBM 可以随时对本出版物中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对任何非 IBM Web 站点的引用都只是为了方便起见才提供的， 不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。本程序的被许可方如果要了解有关程序的信息以达到如下目的: (i) 使其能够在独立创建的程序和其它程序（包括本程序）之间进行信息交换，以及 (ii) 使其能够对已经交换的信息进行相互使用，请与下列地址联系:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

只要遵守适当的条款和条件，包括某些情形下的一定数量的付费，就可获得这方面的信息。

本文档中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获取的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量可能是通过推算估计出来的。实际结果可能会不同。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应者、其出版说明或其他可公开获得的资料中获取。

IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

除其所表示的目标和主题外，所有关于 IBM 未来发展方向和意图的声明，如有更改或撤销，恕不另行通知。

所有 IBM 的价格均是 IBM 当前的建议零售价，可随时更改而不另行通知。经销商的报价可能会不同。

本信息仅用于规划的目的。在所述产品可用之前，此处的信息可能会更改。

本信息包含日常业务运营中使用的数据与报告的示例。为了尽可能完整地说明这些示例，这些示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称均属虚构，若与实际企业使用的名称和地址有任何雷同，纯属巧合。

版权许可证:

本信息包含源语言形式的样本应用程序，用以阐明在不同操作平台上的编程技术。如果是以按照在编写样本程序的操作平台上的应用程序编程接口 (API) 进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无需向 IBM 付费。这些示例尚未在所有条件下经过全面测试。因此，IBM 不能保证或暗示这些程序的可靠性、可维护性或功能。这些实例程序“按现状”提供，不附有任何种类的保证。IBM 对于使用这些样本程序所造成的损害不应承担任何责任。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

如果您正在查看本信息的软拷贝，图片和彩色图例可能无法显示。

商标

IBM、IBM 徽标和 ibm.com 是 International Business Machines Corp. 在全球许多管辖区域注册的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。当前的 IBM 商标列表，可从位于 http://www.ibm.com/legal/us/en/copytrade.shtml 的 Web 站点上的“版权和商标信息”部分获取。

其他名称可能是它们各自的负责人的商标。其他公司、产品和服务名称可能是其他公司的商标或服务标记。

Printed in China