# 定性分析辅助软件——Qualrus[1]简介

夏传玲
中国社会科学院社会学所

在一个典型的定性研究当中，研究人员将通过参与观察、深度访谈等方式，获得大量的录音、录像、档案、图片、访谈笔记等原始材料。为了更加快捷、稳定、透明地分析这些资料，我们需要借助于计算机辅助分析软件。Qualrus 是众多辅助分析软件中，比较具有特色、且对中文具有较强兼容能力的一种[2]。因此，依据该软件的《用户手册》(Brent et al. 2002)和在线帮助文件，本文对其特点作一简要介绍，以帮助初学者熟练掌握这一重要的研究工具。读者也可以参见 Blanco 更简明的介绍(Blanco 2003)。

## 一、Qualrus 的主要特色

Qualrus 把定性分析分解为不同的要素，包括：

(1) 项目（project）：包括研究中的所有数据资料、分析结果和最终报告。

(2) 源文件（source）：包括访谈、观察等方式得来的文本、图片、声音和视频资料。

(3) 段落（segment）：它是源文件分解而来的更小的语义单位，可以是自然段落，也可以一段引文，也可以是一两个词汇，无论长短，段落都是用来显示一个特定语境下的概念（代码）和概念之间的联系（链接）。所谓“扎根”，就是说，概念在段落中具有实例(Glaser & Strauss 1967)。段落体现了概念的指征意义(Kaplan 1964)。

(4) 代码（code）：它是描述现象的最小范畴，是抽象概念的较为具体的表现。各种代码一起，形成对段落的刻划和阐释。代码是在分析过程逐渐形成的，是扎根理论的基石。 同时，它也是逐渐成形的理论框架的基础。代码具有属性，属性具有不同取值。下级代码可以继承上级代码的属性和取值。

(5) 链接（link）：代码之间的联系类型，例如，隶属关系、指代关系等等。在 Qualrus 中，每一种链接都具有指向、对称、传递性和预测性等属性。Qualrus 默认的三种链接类型是“隶属”（isa）、“互斥”（excludes）和“因果”（causes）。研究者可以建立自己的链接类型。链接实例（link instance）是链接类型在代码之间具体实现。彼此用链接相连的代码构成一个“语义网络”（semantic network），它体现了概念的系统意义(Kaplan 1964)。

[1] 以下讨论基于 Qualrus 第 2.1 版。详细信息请访问其主页 www.ideaworks.com/qualrus/index.html 。
[2] 参见不同定性分析辅助软件的比较报告。

(6) 列表（list）：分析的结果常常以列表的形式出现，列表就是具有相同属性的对象集合。例如，含相同代码的所有段落。

(7) 示意图（view）：它是代码及其链接的形象表达。

(8) 报告（report）：上述所有要素都可以存储在不同的报告中，借助于代码、链接、段落阐释等要素，最终达到对社会现象的阐释或解释。任何分析结果都可以存储在报告中。一个项目中可以存储多份报告，报告中的不同成分可以随时删除。

我们可以把这些要素看作是一个不断把原始素材精炼提升的抽象过程，达到“由表及里、去伪存真”的目的。这个抽象过程在 Qualrus 中主要分为三个阶段，即划分段落和编码（形成概念阶段）、概念提炼和理论构建过程以及生成报告阶段。在这三个相互交叉的阶段中，最繁重的任务是编码（coding）阶段。编码是生成代码并把代码分配给段落的过程，它定性研究中最耗时、也最困难。

作为定性分析的辅助工具，Qualrus 的主要特色表现在：

1、可以处理各种类型的原始素材

Qualrus 对定性分析的源文件格式没有多少限制，文件格式可以是图片、语音、视频和文字。支持的文件格式：文本（*.TXT）、格式文本(*.RTF)、视频(*.AVI)、声音(*.WAV、*.MP3)和图片(*.BMP、*.JPG)。在编码和分析的同时，Qualrus 还可以编辑文本文件，例如，修改里面的错别字。

2、可以用图形的方式显示代码及其关系

3、具有一定的智能性

Qualrus 具有依据案例推理（case-based reasoning）、理解自然语言（主要是英语）、机器学习（machine learning）和语义网络（semantic networks）等功能，在编码阶段，它可以向研究者推荐代码。

4、可以辅助理论构建

在写作报告阶段，Qualrus 还有一些理论构建工具，包括归类、代码之间的理论和经验关系、概念的提升和精炼、假设检验等等。

5、减轻编码负担

Qualrus 区分代码和代码属性。有了代码的属性值（attribute-value）之后，研究者就可以视情形而在代码和代码属性之间之间进行选择。对于相同的段落，我们可以有两种编码策略的选择，一是编很多代码，每个代码均没有属性值；二是编较少代码，但每个代码均具有不同属性值。如果子项很多，或者可能的取值很稀少，也可以应用属性值列表。如果一个概念具有不同的属性，特别是当属性的数量很多，或者属性的多寡不能事先确定时，与其把这些属性编作不同的代码，就不如作为代码的属性。例如，年龄的属性可以从

0-100 岁，或者是婴儿、儿童、少年、青年、成年、老年等。另外，把属性用代码表示，还可以造成指代上的混乱。同时，属性赋值会增加 Qualrus 的脚本处理能力。这一设计，加上智能辅助，Qualrus 给编码过程带来简便、快捷和稳定等优势。

6、增加研究的形象化

任何检索结果、分析结果都可以用文字和示意图两种形式表达。

7、强大的分析语言

Qualrus 拥有自己的脚本语言（Scripts），它是完成特定任务（例如，检索）的计算机语言指令。实际上，任何任务（包括通过菜单和工具按钮来操作的任务）都可以用脚本来完成。脚本语言增加了 Qualrus 的适应性、灵活性和开放性，从这个意义上讲，Qualrus 可以看作是一种专门处理文本的开放语义网络平台。

8、丰富的辅助工具

除了脚本之外，Qualrus 还有一些常用工具，例如，给不同段落划归较大类别的归类工具（categorizing tool）、将一个概念分拆为不同概念的概念精炼工具（concept refinement tool）、将不同概念合并成一个更宽泛的概念的概念抽象工具（concept generalization tool）和用来考察语义网络、直接链接或间接链接的网络工具（network tool）。这些工具在理论构建中的作用，我们将在后面详述。

9、团队工作

Qualrus 的输出和输入功能、编码员及其编码的记录功能、登录密码等等，都给团队协作奠定了基础。一个项目中的所有文件都可以输出给另外一个人使用。其他项目中的代码、脚本和链接类型也可以输入到自己的研究项目中。有了这个特点，研究者之间就有了两条联系通道：共享源文件和共享分析框架。这样，每一个新项目就没有必要一切都从零开始。

## 二、Qualrus 辅助的编码过程

1、形成编码框（Coding Scheme）：

理论的特征是洞见、有效和强大。但是，从繁杂的经验材料到系统的理论框架，中间需要很多的步骤，其中，最重要的一步就是建立编码框。

从前面的介绍可知，段落是源文件中的一个连续部分，包括词汇、句子和段落，用以进行编码。在 Qualrus 中，段落可以相互重叠。代码是段落中所反映出的基本概念。代码通过链接（links）相互关联，连接的类型有逻辑关系、因果关系或相关关系等。一个项目中的所有代码和链接，则构成“编码框”（coding scheme），这是定性研究中的理论原型。

编码框是一个定性分析项目背后的理论雏形，源文件是形成和检验这种理论雏形的数据之一。编码是把这两个方面联系起来的途径。

和扎根理论(Glaser & Strauss 1967; Strauss & Corbin 1998)所暗示的经验主义取向不同，在 Qualrus 的架构中，编码框独立于源文件。这意味着，编码框的来源是多样的。例如，我们可以借用其他相同主题研究中的代码框，通过 Qualrus 的输入（import）功能，获得最初的代码及其链接。也可以从现有的理论中抽取概念，直接生成新代码。

在这些基础之上，我们再对经验材料进行手工编码的操作。编码的原则是先划分段落再生成和分配代码，即先标记文本的特定段落，如访谈材料中的一段文字，然后从中形成代码，并分配给这一段落。一开始，多数新代码都是描述性的，例如，指明所发生的事情，采访的日期，被访者的性别和住址，该段落是被访者主动披露的还是对研究者提问的回答，采访时是有其他人在场还是和研究者一对一的环境中，等等。再往后，我们将提炼更多的分析性代码，用来探讨所感兴趣的现象及研究问题。

2、生成新代码

在 Qualrus 中，创建新代码的操作在“代码编辑器”中完成。在建立新代码的同时，尽可能建立代码之间的链接。这是编码的第二个原则。因为这些链接有助于厘清代码之间可能的关系，如因果、序列、关联、隶属、网络等等，也有助于 Qualrus 发挥其智能辅助的功能。

Qualrus 的一个特色是让代码自身具有很多属性，这些属性被称之为“代码细节”，我们可以在“代码编辑器”中对它们进行编辑。代码细节包括“代码名称”、“冠词”（代码名称之前的词汇）、“详细描述”（可以是代码定义或对代码的描述）、“近义词”（相同代码的不同词汇或词组）、“属性及其赋值”（表示代码的特征）。Qualrus 的智能功能，例如，自动编码、编码提示等，和代码名及其近义词息息相关。如果一个段落中出现代码名及其近义词，则 Qualrus 会给出编码提示。Qualrus 的某些智能功能，虽然是建立在英语之上，大多不适用于中文，例如，“冠词”属性；但有些功能（例如，近义词）还是可以应用于中文环境。

3、简化编码流程

Qualrus 主要通过四种策略来简化编码过程。一是通过脚本语言，自动划分段落并进行自动编码，我们将在下面的脚本语言一节作更详细的介绍。二是推荐代码，在简化编码任务的同时，增加编码的前后一致性。三是快捷操作方式。四是集团操作。

集团操作主要通过检索和文栈（stacks）两条途径实现。在 Qualrus 的 QTools 中，有专门的检索工具（Search），可以用来检索段落、代码、源文件、段落或句子中的特征，检索的条件是代码和文本及其逻辑组合。符合检索条件的段落会出现在结果窗口中，我们可

以对它们进行集团操作，同时为这些段落添加或删除代码。另一个工具是段落归类（Categorize），用来把段落划分为少数几个类别。具体操作是，先建立不同的文栈，然后把标志着段落的页面图标从工作面（workspace）窗口拖放到文栈图标上。单击文栈图标，就会在当前文栈窗口显示该文栈当中所有的段落。在拖放的同时按下“Ctrl”键，就是把一条段落移动到文栈当中，并从工作面窗口消失。这样，相同的段落就不会出现在不同的文栈当中。段落可以在工作面窗口和文栈窗口之间相互移动。当分类完成之后，我们就可以对同一文栈中的所有段落进行编码，或者把代码从这些段落删除，从而完成集团操作。

在快捷操作上，Qualrus 主要的功夫花在三个方面。在划分段落时，除了左键标记外，Qualrus 还设计了一些快捷操作，例如，双击左键选择单词，三击左键选择整个自然段落。同时，它还具有浏览编码段落的快捷键（上一个、下一个、第一个和最后一个段落，图标分别是<、>、<<和>>）。图形的段落是图片中的一个长方形区域。视频的段落是以起始频帧和中止频帧来界定，Qualrus 有两个按钮，分别标识起始频帧和中止频帧。当频帧以 15-30 帧/秒的速度播放时，我们将可以看到连续的图片。声音文件则分成很小的时间单位，语音片断是以起始时间和中止时间来界定的，标识方式和视频相似。

高亮段落后，右键菜单出现编码选项。在编码窗口中，会出现三个代码列表，左下方是所有代码的列表，右上方是已经赋值给这个段落的代码列表，左上方是 Qualrus 推荐的代码列表。双击左边的代码列表，就可以把左边的代码赋值到右边，双击右边代码则是删除所赋值的代码。在编码窗口中，还有生成新代码、便签、依据案例推理和“Q 编码”的快捷按钮。窗口右下方是自动生成的代码小结，包括代码的定义、属性和正在编码的段落。其中，代码属性是可以编辑的。

在生成新代码时，Qualrus 有图标、右键等方式，方便编码工作。除此之外，我们也可以用克隆的方式，复制已有的代码，生成新代码。克隆代码和原代码之间只有代码名的不同，其他的代码细节均相同。

当有成千上万个段落，上百个代码时，编码任务就变得十分纷繁单调。随着项目的进行，代码越来越多，编码任务也就越来越繁重。因此，迫切需要能够简化编码任务。Qualrus 采用的解决方案是智能编码，包括（1）迷你专家系统或脚本语言，用来发现大量资料中的普遍模式；（2）机器学习策略，用来识别额外的模式；（3）依据案例推理，用来识别相似个案、关联或概念链接；（4）单词或短语的自然语言识别。

Qualrus 把自己的智能计算程序称之为“QCode”。当我们在源文件中标识一个段落时，Qualrus 会先把这一段落送到一段脚本中进行分析，然后把结果显示在编码窗口中的左上角，作为程序所推荐的代码。研究者可以修改这些代码，QCode 窗口左边的脚本是当前使

用的脚本[1]，右边的“更多脚本”是没有使用的脚本，两者之间可以用箭头图标相互移动。具体脚本的内容则可以从主窗口中的脚本树窗口中查看。右键单击编码窗口中的推荐代码，就会出现编码助理窗口，里面显示了每一个推荐代码的理由。例如，由同义词推荐。

（1）嵌入专家系统

Qualrus 具有内建的专家系统进行推理，例如，“if X then Y”。使用的规则包括因果联系（X 是 Y 的因，因此，当 X 出现时，Y 应当出现），关联原则（X 和 Y 关联，则 X 的出现意味着 Y 的出现），这是所谓“的预测性规则”。在 Qualrus 中，所有预测性的代码链接都将造成推荐代码的出现。

Qualrus 的第二类推荐规则是基于经验关联，例如，两个代码之间的“同现”（co-occurrence）、序列或语境。如果多个代码之间经常一起出现在段落中，它们之间的关系就是“同现”关系。当多个代码总是出现在一些代码的前一个段落中，则它们之间构成“序列”关系（sequence），即依据前一个段落中的代码，预测后一个段落中的代码。语境规则是指相同语境的相邻段落，这些语境并没有反映在文本中，但是所有相邻段落共有的。但代码之间具有层序链接时，语境规则也适用。

（2）机器学习

Qualrus 检测每一个脚本的绩效。每当一个脚本所推荐的代码被研究者接纳时，这个脚本的成功分就增加一分，否则，失败分就增高一分。当成功和失败的比例低于一个阈限时，这个脚本就处于休眠状态，不再向研究者推荐代码。

Qualrus 的第二种学习方式是基于代码之间的同现率。它计算任何一个代码和其他代码之间同现的条件概率，例如，如果代码 X 出现在 12 个段落中，其中的 10 个段落中，代码 Y 也出现了，则给定 X 出现，Y 代码也出现的条件概率就是 10/12，即 0.83。Qualrus 默认的条件概率阈限是 0.76，最小的段落数是 52，当这两个条件符合时，它就将依据代码 X 而推荐代码 Y。

这一机器学习策略是 Glaser 和 Strauss 的“经常比较方法”(Glaser & Strauss 1967)、Ragin 的“比较法”(Ragin 1987)以及 Mill 所说的“异同法”(Mill 1872)的计算机实现。

（3）理解自然语言

让计算机理解自然语言是一件比较困难的任务。Qualrus 利用关联语义网络在理论语境下进行推理，即试图把握概念的系统意义。每一个概念具有两类意义，一是系统意义（systemic meaning），它来自和其他概念的逻辑联系，包括因果、层序等；二是指征意义（referential meaning），它是概念的经验指标，包括可观察的属性和品质(Kaplan 1964)。

[1] 在 Qcode 中的脚本，必须具有特定格式，例如，必须运行于 CurrentSource 和 CurrentSegment 两个变量上。

[2] 在菜单 Tools/Option/Learning 项下，更改默认值。

常见的概念链接有层序链接，例如，“部分关系”（is_a 或 part_of），因果链接，关联链接。每一种链接均具有自己的指向性、对称性、传递性，以及是否为一对一、一对多、多对一及多对多的关系。在主轴编码中，代码之间的关系是由研究者设定的。因此，Qualrus 的关联网络是主轴编码的实现(Glaser & Strauss 1967; Strauss & Corbin 1998)。

另一个特征是利用同义词或近义词的功能。这一策略相对简单，但很实用，也比较适合于中文的处理。

（4）依据案例推理

依据案例推理即是通过记忆过去的案例的一种推理策略，当它面临一个难题时，它将从记忆库中寻找最相近的个案，加以改进，以适应新的情形(Leake 1996)。Qualrus 利用依据案例推理，把当前的段落和以前的编码段落相比较，然后推荐代码。所涉及到的两个主要脚本分别是“Suggest Codes of Segments Similar by Codes”（依据代码相似度推荐段落代码）和“Suggest Codes of Segments Similar by Words”（依据词汇相似度推荐段落代码）。

## 三、Qualrus 辅助的概念形成过程

得到代码，并不意味着定性分析的结束，从某种意义上讲，这仅仅是定性分析的开始。因为编码过程也可以看作是一个通过比较，不断对资料进行分类的过程。从代码到概念，还有一个提炼的过程。

在辅助概念形成的过程上，Qualrus 也有相对较高的比较优势，表现在代码、链接、便笺、段落阐释、归类、概念的合并和拆分、质量控制和辅助工具等方面。

1、代码

Qualrus 具有生成代码小结的功能，包含代码的定义、代码和其他代码之间的链接等等。这样，我们就比较直观地看到代码后面的理论概念的指征意义和系统意义，从而对概念有更深的把握。Qualrus 利用系统意义不断生成描述代码和其他代码之间关系的英文句子，它有助于避免“代码漂移”的现象出现，即代码的使用方式在研究过程的前后不统一。

2、链接

Qualrus 允许研究者建立自己的链接类型，用来表示不同代码之间的关系。例如，“isa”是特例关系，“part of”是整体和部分的关系，“causes”是因果关系。

如果是我们自己添加的新链接类型，那么，我们就需要指明这种类型的几个属性，包括链接名、指向（包括环状和非环状两个值，关联就是一种环状指向，特例关系则是非环状的）、对称（是指链接的方向性，包括对称和非对称两种）、传递性（是指关系在多个链

接之间的传递，有传递和非传递两种）和预测性（代码 A 的出现和代码 B 之间的关系，A 出现则 B 一定出现是“预测”，A 出现则 B 一定不出现是“排除”，这两种情形之外，则无预测关系）。这些属性决定了链接在语义网络图中的作用。

除了上述属性之外，链接还具有“默认动词”，从而对代码之间的关系作一个初步描述。一种方式是主动式（SVO），即主语——谓语——宾语的格式，另一种是被动式（OVS），即宾语——谓语——主语的格式。

3、便签

在 Qualrus 中的所有对象上，研究者都可以附加笔记或便签，包括文本源、每一个段落、每一个代码、每一个链接、每一个图示以及每一个脚本。添加便签的方式是右键菜单。便签中的内容可以被检索、查看、写入报告、进行编辑等等操作。便笺是研究者在编码过程中的所有感想，是从理论概念到经验指标、从经验指标再回到理论概念的认识过程的纪录。这种纪录会增加研究过程的透明度，它不仅是教学的好材料，也是同行监督的重要依据。

4、段落阐释

Qualrus 可以利用语义网络，自动生成段落的理论阐释。在高亮的段落上右键选择“段落解释”就可以得到代码之间关系的语义网络，以及本段落的代码以及自动生成的文本描述，包括本段落所使用代码的描述，和这些代码有理论联系的其他代码，预见一些具有更进一步理论联系的代码。通过排列代码的指征意义和系统意义，它有助于研究者从理论上把握段落的语境，让一个代码和其他代码之间的隐含关系直观地显示出来，从而判断所编代码是否恰当，或者是否需要添加新的代码。这样，编码的信度和效度也会有所提高。这种自动生成的理论解释也可以用于评估理论，如果理论解释常常不能令人满意，这是理论需要修正的征兆。这样，编码和理论构建之间就会相互强化，理论指导编码过程，而代码也可以在理论的语境下加以理解。

同时，这些段落阐释还可以帮助研究者确定编码的恰当性。如果代码之间的联系越多，这种阐释的理论含义就越强。这样，理论框架和代码过程就可以进行互动。

5、归类

Qualrus 用归类工具帮助研究者来整理已编码的段落，在代码之上形成更宽泛的类别。这一工具可以把各种段落形象地归成一摞一摞（stack）[1]，然后，给每一摞段落加上或删除相同的代码。归类既是一种简化编码负担的操作，也是形成概念雏形的一种手段。

6、合并和拆分

概念的概括和精炼，前者是多个概念合并成一个更抽象的概念，后者是一个概念拆分成不同的概念。Qualrus 从两个代码和其他代码之间的分析关系、两者同时出现在相同段

落的频率，来判断两个代码合并的可能性。经常同时出现在一个段落中，且具有相似代码链接的两个代码，可能是一个更宽泛概念的示例，应当考虑合并。依据这两个标准，Qualrus 将给代码对标出一个建议合并的分值。

分拆概念、调整概念之间的链接等是精炼概念的常见方式。Qualrus 精炼概念的逻辑是，如果一个代码和其他代码之间具有相对较少的联系，则这个代码可能尚没有得到明晰。Qualrus 使用的统计量是卡帕值，它一个具有误差比例减少性质的统计量，计算公式是：

$$\text{卡帕值} = \frac{\text{观察一致} - \text{偶然一致}}{1 - \text{偶然一致}}$$

如果两个代码之间的关系较弱，其卡帕值就较小。

7、区分流派

Qualrus 默认的段落描述是包括所有知识库中的代码及其联系，也可以通过“流派”选择某个代码，这样，和这个代码有直接或间接联系的代码才会出现在段落描述中。这些代码可以是一个特定理论、特定事件、特定行动者、特定行动等等。

8、质量控制

Qualrus 保留了所有对象的细节和历史，这样，就可以对编码进行质量控制。例如，在编码窗口中，右键菜单中选择“细节”，就会显示原始文献的加入日期，加入者和到目前为止所编过的代码总数，右键菜单中选择“历史”，则显示编码者、操作、操作时间和所赋值或编码。在代码上右键选择“细节”，则出现编码细节，包括文本段落，编码者、编码时间、编码原因。右键菜单中选择“历史”，则显示编码者、操作、操作时间和所赋值或编码。质量控制增加了定性研究过程的可问责性。

9、辅助工具

在 QTools 中，还有一些工具能够辅助概念的形成。例如，代码统计工具（Statistics）会显示代码或代码对的出现频次。这样，我们就会对研究项目中的理论抽样(Glaser & Strauss 1967; Strauss & Corbin 1998)状况有一个直观、量化的评估。代码偶遇工具（Coincidental Code）可以检查代码在相同段落中偶遇的概率，发现潜在的代码之间的关系。而且，Qualrus 通过自己推荐的代码，来监测编码者的编码决策，测量编码者和软件之间的一致性。同时，它还有一个专门的工具（QTools 中的 Consistency）检验不同编码者之间的一致性。

[1] 拖拽操作是拷贝，Ctrl＋拖拽操作是移动。

## 四、理论构建和理论检验

理论构建是一个发现之旅，任何工具也无法替代思维的洞见。不过，Qualrus 可以在从代码上升到概念的基础上，借助于语义网络，给研究者较大的帮助。这些帮助体现在（1）搜寻模式；（2）理论饱和；（3）理论模块（Theory Modules）；（4）假设检验；（5）依据案例推理和（6）理论验证。

（1）搜寻模式

Qualrus 拥有两类工具，可以帮助研究者构建和检验理论。一是 QTools，二是图示工具。其中，QTools 包括检索、假设检验、形成范畴、代码同现、精炼、概括、统计和一致性等工具。借助于它们，Qualrus 采用不断比较、依据案例推理和自动生成的文本解释等三种方式来寻找资料中的模式。例如，Qualrus 的“同现代码”（Coincidental Codes）工具能够然研究者考察任何两个代码之间在段落出同时出现的模式。结果出现在一个表格中，单击 X、Y、Not X 和 Not Y 四个格子，就会显示相应的编码段落。这种不断在资料中搜寻模式的能力，既可以促进研究者的理论敏感性，也可以让不同的理论洞见和资料之间形成对话关系。

（2）理论饱和

当增加新个案不需要调整理论来解释新个案时，就出现了“理论饱和”的现象(Glaser & Strauss 1967; Strauss & Corbin 1998)。Qualrus 检测研究者的编码过程，看看添加新的编码段落是否带来理论的修正，如果连续出现 n 个编码段落，理论还没有修正的情形，它会提示研究者考虑是否出现理论饱和的情形。

（3）理论模块

所谓“理论模块”，即是能够把握较大理论或模型中的一个重要方面的、规模适中的一组命题。Qualrus 可以帮助研究者形成理论模块，借助于程序规则、问题框架和语义网络等工具，定性分析者就更方面、系统地就理论的某个方面加以研究。

（4）假设检验

Qualrus 使用假设检验（Hypothesis Testing）来帮助研究者来精炼自己的理论。每一个假设检验具有自己的名称，If 和 Then 语句中，可以包括代码、文本或者是与某个代码关联的代码，符合假设的编码段落将出现在结果中。但需要注意的是，Qualrus 的假设检验遵循的是定量分析的逻辑，即以代码在编码段落中出现的共同出现的频率来判定假设的真伪。这一点是值得商榷的。

（5）依据案例推理

在 Qualrus 的编码窗口有一个依据案例推理的按钮，可以用来设定 Qualrus 的个案（个

案单位是段落）推理参数，包括适用范围，文本和代码的相对重要性以及选择匹配个案的阈限。在编码时，单击“依据案例推理”按钮将使得 Qualrus 考察所有已编码的段落，尝试发现相似的个案，并显示其内容、所含代码、以及考虑到差异后软件所推荐的代码。

借助于智能工具，Qualrus 将随着项目的进展，而变得越来越聪明。同时，它利用图示动态地表达逐渐成形的扎根理论。

（6）理论验证

思考其他可能的解释，验证自己的结论，是定性研究中的一个重要阶段。这个阶段包括评估理论的逻辑一致性、评估基本假定、寻找否定证据、识别缺失的部分、评估并确保信度、以及确保发现的代表性。Qualrus 可以从理论中作出自己的推理，这不同于研究者自己的理论推理，因而给理论的逻辑一致性提供了一个安全阀。这个角色表现在：

- 避免同义反复：Qualrus 区分机器生成的代码和研究者自己的编码，因此，在检验理论时，就不会用理论所推理的个案来检验理论。
- 识别和检验基本假定：Qualrus 可以识别出在理论中扮演重要角色的概念，追索理论逻辑及其含义，帮助研究者澄清理论的基本范围条件。已编码的段落是 Qualrus 的检验库，如果理论发生变化，Qualrus 可以重新在这些检验库上再运行一次，并比较两者之间的异同。
- 发现否定证据：Qualrus 的工具可以鉴别出和特定假设不同的段落或代码之间的关系。
- 发现缺失和代表性：统计工具可以帮助研究者发现没有充分表达的代码，厘清理论的范围条件。
- 提高可重复性：Qualrus 形成的动态、互动性数据库，可以让其他研究者用来分析相似的数据，从而增加了研究的可重复性。

## 五、存储列表和报告结果

Qualrus 不仅可以简化编码流程，辅助概念提炼和理论构建的过程，而且，在定性研究的最后一个阶段：报告写作的过程中，起一个补充的角色。在 Qualrus 中，承担这一角色的两个主要工具是列表和报告。

1、列表

Qualrus 允许研究者存储检索等操作所产生的列表，或者把前者所生成的列表作为后续操作的输入。这样，它就可以辅助研究者完成比较复杂的任务。

下面的一个脚本生成并存在列表：

List(人物).{生成列表“人物”}
If Project.GetCode(“社会学家”,人物){把代码“社会学家”列入“人物”列表中}
Then
社会学家.GetDescendants(人物)
DisplayList(人物)
Endif.

生成的列表可以在列表编辑器中编辑，如，添加或删减列表元素，改变元素的顺序等。Qualrus 区分对象列表和对象名列表，当在 Project.Lists 中存储一个列表、或在列表编辑器中创建一个列表时，我们得到的是对象名列表。当我们在脚本中生成一个列表，我们得到的是对象列表。

2、报告

在分析过程中，我们可以把各种对象存储在报告中，然后用报告编辑器（Report Editor）生成或编辑报告，文件格式是超文本格式。借助于这一工具，我们把握分析结果的能力和效率都有很大的提高。

这些列表和报告都可以作为研究报告中的写作素材，帮助我们把抽象的理论概念和生动的经验现实之间搭了一个桥梁，达到对研究对象进行深描的目的(格尔兹 1999)。

## 六、Qualrus 的脚本语言

Qualrus 中的“脚本”是用来完成特定任务的一系列计算机指令。它可以在脚本编辑器中编辑和运行，类似于 Perl 语言。在项目文件中，存储项目中所有脚本命令的文件名，以 IQI 后缀结尾。一个脚本可以调用其他脚本，并相互传递信息。脚本语句分为查询语句和执行语句，前者不改变 Qualrus 内部的数据库结构，只把查询结果返回到脚本编辑器的结果显示窗口，后者则改变数据库的内容。例如，下列查询语句返回当前项目中的所有代码：

List Project.Codes.

每一个脚本可以包含多个语句，每一条语句以句号结尾。Qualrus 的脚本语言允许使用注释语句，以｛｝表示，可以放在行末或。控制结构是一种特殊的语句，在其内部可以包含其他语句，且不使用句号。例如，

```
脚本="我是脚本". {句号位于末尾}
if 脚本="我是脚本" then
  脚本="我比较牛" {嵌套语句不使用句号}
endif. {外层语句使用句号}
```

Qualrus 是一个以对象语言，主要的对象层次，从大到小，依次是项目、源文件、段落、代码和代码属性。

## （一）变量

Qualrus 的脚本语句具有四种变量：字符串、数字、布尔代数和列表。对象名可以是英文字母或中文，不区分大小写，但不可以用 Qualrus 内部的关键词，例如，CODE、CODES、FIND、LINK、LINKS、PROJECT、SEGMENT、SEGMENTS、SOURCE、SOURCES、WHEN 和 WHERE。和其他编程语言不同的是，任何对象都无需预先定义，就可以直接使用。

字符串由字符构成，以双引号界定。字符串的合并运算符是加号，例如，

```
姓="夏".
名="天长".
姓名=姓+名.
writeln(姓名).
```

得到“夏天长”这个新字符串。

数字包括整数和实数，加减乘除运算符分别是+、-、*和/。Qualrus 没有转换字符串和数字的函数，一个技巧是相互赋值，例如：

```
number(数字).
字符="6.6".
数字=字符.{赋值运算}
数字=数字+0.026. {算术运算}
writeln(数字).) {显示结果}
```

就将字符串 6.6 转换成数字 6.626，并存储在变量“数字”中。

布尔代数只有 TRUE 和 FALSE 两个值，只有一种运算即 not（非），例如：

```
完成=TRUE.
未完成=not(完成).
writeln(未完成).{显示结果为“FALSE”}
```

在条件语句中，还有“<>”表示不等于，“=“表示等于，“>”表示大于、“<”表示小于等、“<=”表示小于等于、“>=”表示大于等于。

列表是 Qualrus 中的对象，每一种对象具有自己的 methods（方法）和 properties（属性）。方法是对象的行为方式，属性是对象的特征。例如，Professor.WriteGrant 是指明对象 Professor 采用 WriteGrant 的方式相应脚本的指令。

定义这四种变量类型的函数分别是 String()、Number()、Boolean()和 List()。除此之外，Qualrus 中还有整数 Integer()、双字节数 Double()等较不常见的类型。

**（二）基本函数**

Qualrus 以函数的方式进行一些基本操作，包括输入和输出，字符串处理等。

1、输入和输出

Qualrus 用 display()和 writeln()两个指令进行输出操作，参数为字符串或字符变量。两者的不同之处是，writeln 是逐行显示，而 display 不分行。在字符串中可以使用格式<B></B>（加粗）等格式标记。例如，

display("<B>这几个字的字体将加粗。</B>"). {显示**“这几个字的字体将加粗。”**}

此外，Qualrus 还有一些专用的显示命令。例如，DisplayHierarchy(链接实例)或 DisplayTree(链接实例,代码名)，它们将以层状或树状显示链接。DisplayPaths(路径变量)则显示代码之间的路径图。List

对于数字变量，我们可以设定其输出格式。指令 Format(变量名,2,3)有三个参数，第一个是数字变量名，第二个参数是位数，第三个是小数点后面的位数。例如，

Double(A). {A 是双字节数，注意此时 A 不能用中文表示！}

A=1232.3346212. {A 赋值}

display(A). {显示 1232.3346212}

A=Format(A,2,3).

display(A).{显示 1232.335}

2、字符串的处理

Qualrus 有很多对字符串处理的专用函数，作为特定对象的专用方法（参见下一节）。这里只涉及通用的函数，包括字符替换 StringReplace()、字符分割 Split("字符串"或字符变量, "分隔符", 结果列表)、字符长度 Length("字符串"或字符变量)、字符计数

SubstringCount("字符","字符串"或字符变量)。

“+”运算将两个字符串合并成一个。

（1）字符替换

String(称谓).

称谓="先生!".

改谓=StringReplace(称谓,"先","后").

Display(改谓). {结果显示“后生!”}

（2）字符分割

Split("字符串"或字符变量, "分隔符", 结果列表)，将字符串或字符变量中的字符，用分隔符进行分割。例如，

aString="第一、第二、第三第四".

Split(aString, "、", aList).

display(aList). {显示“第一
第二
第三第四”}

（3）字符串长度

Length("字符串"或字符变量)将返回字符串或字符变量的字节长度，注意汉字的一个字为两个字节。例如，

i=Length("字符串 1").

display(i). {显示结果为 7，其中“字符串”三个字为 6，数字 1 的长度为 1}

（4）字符计数

SubstringCount("字符","字符串"或字符变量)，它有两个参数，第一个是待计数的字符，第二个查找范围，返回在“字符”在字符串中的出现次数，如果没有，则返回零。

（5）字符位置及其指针

Pos、Position 和 IndexedPosition 可用来确定某个字符在字符串中的位置。Brent 等人建议不要用 Pos 指令，而用 Position("字符","字符串"或字符变量)、IndexedPosition("字符","字符串"或字符变量,i)，其中第三个参数表示计算第 i 个出现的字符(Brent et al.

2002)。如果该字符在字符串中不存在，返回零。例如，

名句="风声雨声声声入耳".

n=SubStringCount("声",名句).

i=Position("声",名句).

j=IndexedPosition("声",名句,1).

k=IndexedPosition("声",名句,4).

m=Position("国",名句).

Display(i).{显示 3}

Display(j).{显示 3}

Display(k).{显示 11}

Display(m).{显示 0}

Display(n).{显示 4}

（6）子字符串函数

IsSubstring()用于判定某个字符是否为一个字符串的部分。它需要两个参数，第一个参数是待判定的子字符，第二个参数是字符串，返回值是布尔值，真值为 TRUE，假为 FALSE。例如，

名句="风声雨声声声入耳".

if IsSubstring("声",名句) then

display(名句) {显示 “风声雨声声声入耳”}

endif.

3、路径处理

两个代码之间的一系列链接，构成它们之间的一个“路径”（Path）。处理路径的 Qualrus 函数是 GetPaths()，它需要六个参数，第一个参数是代码范围，第二个参数是链接类型，第三个参数是选项，包括 BOTH、OR、AND 或省略等，这是一个可选的参数，第四个参数是起点代码，第五个参数是终点代码，最后一个参数是一个包含所有路径的对象变量。例如，

{GetPaths}

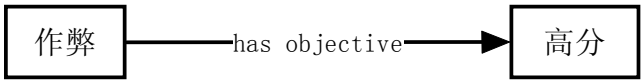
代码=Project.Codes.

If GetPaths(代码,"has objective,causes",BOTH,"作弊","高分",Paths)

then DisplayPaths(Paths) {显示“作弊”和“高分”两个代码之间的“目的”和“因果”连种类型之间的一个链接，单击则显示一个示意图，参见图 1}

```
else Writeln("no paths found")
endif.
```

图 1 代码之间的路径示意图




4、函数调用

其他计算机语言的函数调用（CALL），在 Qualrus 中是用三个指令完成的，它们分别是 Input()、Script()和 Return()。

被调用的脚本无法输出结果（包括 writeln()等指令的屏幕显示），这就需要调用脚本提供一个返回列表的参数，存储被调用脚本的输出结果。例如，首先编写被调用的脚本，并在脚本编辑器中，把示例脚本内容“Save As”（另存为）“CalledScript”。

```
{CalledScript}
list(ArgsA).
list(ArgsB).
ArgsA.clear.
ArgsB.clear.
input(ArgsA). {从调用函数处得到第一个参数的赋值}
ArgsA.GetString(1,aList). {赋值给 aList}
bList=aList+"已经被调用". {被调用程序的主部分}
ArgsB.Add(bList). {把结果存入第二个参数}
return(ArgsB). {返回第二个参数}
```

调用函数的示例：

```
{CallScript}
list(ArgsA). {声明第一个参数}
list(ArgsB). {声明第二个参数}
ArgsA.Clear. {清空第一个参数}
ArgsA.AddString("调用函数"). {第一个参数赋值}
```

Script("CalledScript",ArgsA,ArgsB). {调用一个脚本，名为 CalledScript，传递的值位于第一个参数 ArgsA 中，返回的结果存储在第二个参数 ArgsB 中}

```
writeln(ArgsB). {结果显示“调用函数已经被调用”}
```

5、控制结构

Qualrus 的控制结构包括条件语句和循环语句。条件语句有一个指令 if-then-else-endif，它内部的语句不需要句号。在条件语句中，then 之后可以跟多个语句，而 else 是可以省略的。例如：

```
                    {条件语句的示例}
幸运数=7.
if (幸运数<5) then
  幸运数=幸运数+1 {第一选项}
else
  幸运数=幸运数-1  {第二选项}
endif.
writeln(幸运数).
```

这一个脚本的运行结果是幸运数等于 6。它的第一个语句把幸运数赋值为 1，然后用条件语句判断这个幸运数是否大于 5，如果大于 5，则减 1，如果小于 5，则加 1。复合条件可以用逻辑运算符（and、or、not，即逻辑和、逻辑或和逻辑否），例如，大于 0 且小于 3 的幸运数，这一条件可以表示为((幸运数<3) and (幸运数>0)。条件相等的符号是“=”，不相等的符号是“<>”。

循环语句重复执行相同的语句，循环的次数可以是固定的，或者由条件语句限定。固定循环是对一个列表中的所有元素重复相同的语句，循环指令是 foreach-in-endfor。例如，

```
{固定循环的示例}
foreach 代码 in Project.Codes
  writeln(代码.Name)
endfor.
```

这一脚本将显示项目中的所有代码。其中，Project.Codes 是项目代码的列表，它将存储在“代码”变量中，代码.Name 是代码对象的一个属性，即代码名称。

条件循环指令是 while-do-endwhile，当 while 语句中的条件得到满足时，脚本就中止循环。例如：

{条件循环的示例}

i=1.

list(社会学家).{生成一个列表变量，名称为“社会学家”}

社会学家.AddString("韦伯").{在其中添加第一个元素，注意 Qualrus 列表元素的指针从 1 开始，AddString 是列表对象的一种方法}

社会学家.AddString("马克思").{在其中添加第二个元素}

社会学家.AddString("涂尔干").{在其中添加第三个元素}

社会学家.AddString("帕森斯").{在其中添加第四个元素}

while (i<=社会学家.Count) do {Count 是列表的一种属性，即元素总数}

社会学家.GetString(i,名单){把“社会学家”列表中的第 i 个元放入变量“名单”中，GetString 是列表对象的一种方法}

writeln(名单)

i=i+1 {注意计数器的递增！}

endwhile.

### （三）主要对象的属性和方法

在 Qualrus 中，主要的对象类型分别是项目(Project)、源文件(Source)、段落(Segment)、代码(Code)、编码(Code Assignment)、链接类型(Link Type)、链接实例(Link Instance)、列表(List)等。下面，我们将分别介绍每一种对象的重要属性及其主要方法。

1、项目

项目对象只有一个名称，即 Project，它具有很多属性，包括 Project.Sources（所有源文件列表）、Project.CurrentSource（当前源文件）、Project.Codes（所有代码）、Project.Memo（所有便笺）、Project.CodesCount（代码总数）、Project.CurrentSource（当前打开的源文件）、Project.Scripts（所有脚本）和 Project.Links（所有链接）等。

我们可以用 Project 所具有的特定方法对源文件、列表、代码和链接四类对象进行操作，包括检索和增减。如果我们需要在项目中添加新的源文件，则可使用 Project.MakeNewSource 方法，它需要两个参数，一个是源文件名（包括路径）、一个是存储源文件名的列表变量，返回一个布尔值，成功添加则取值 TRUE，否则为 FALSE。如果需要删除一个源文件，则可以用 Project.DeleteSource 方法，它需要一个参数，即源文件名，返回一个布尔值，成功删除则取值 TRUE，否则为 FALSE。例如，

{源文件的添加和删除示例}

```
if Project.MakeNewSource("D:\source\Social Change.mp3",源文件) then {以绝对路径的方式表示源文件名}
  display(源文件) {以相对路径的方式显示源文件名，即..\..\Social Change.mp3}
  writeln("删除当前的源文件 "+源文件.Name)
  if  Project.DeleteSource(源文件.Name) then {删除源文件}
    writeln("成功！")
  else
    writeln("失败！")
  endif
else
  writeln("找不到源文件!")
endif.
```

我们也可以Project.FindSource或Project.GetSource两种方法来检索项目中的源文件，这两个指令均需要两个参数，一个是源文件名、一个是存储源文件名的列表变量，返回一个布尔值，成功添加则取值TRUE，否则为FALSE。例如，

{检索源文件示例}

```
If Project.FindSource("Social Change.mp3",源文件) then
  writeln("找到源文件"+源文件.Name+"！")
else
  writeln("找不到源文件！")
endif.
```

项目中的其他方法分别用于检索代码、列表和链接，指令分别是GetCode/FindCode、GetList/FindList和GetLink/FindLink，三者均需要两个参数，一个是对象名、一个是存储对象名的列表变量，返回一个布尔值，成功检索则取值TRUE，否则为FALSE。例如，

{检索项目对象示例}

```
If Project.FindCode("资本",代码) then
  writeln("找到代码"+代码.Name+"！")
else
  writeln("找不到该代码！")
```

```
endif.
If Project.GetLink("隶属",链接) then
  writeln("找到链接"+链接.Name+"！")
else
  writeln("找不到该链接！")
endif.
List(社会学家).
If Project.FindList("社会学家",列表) then
  writeln("找到列表"+列表+"！")
else
  writeln("找不到该列表！")
endif.
```

2、源文件

源文件位于项目之下，是项目的属性之一，即 Project.Source。它具有 Source.Name（源文件名）、Source.Segments（源文件的段落列表）、Source.CurrentSegment（源文件中的当前段落）等属性。

```
Source.SegmentByID(inID) returns object
Source.PrevSegment returns object
Source.IsFirst(in segment object) returns boolean
Source.SeekSegment(direction constant, testScript, VAR in/out segment) returns boolean
Source.SeekSegments(direction constant, testScript, VAR ioList)
```

对于任何一个源文件，我们都可以设定自动编码选项。指令 Source.SetAutocode("TRUE")设定自动编码，Source.SetAutocode("FALSE")取消自动编码。

同时，Source.AutoCodeScripts 可用来调用一系列脚本，进行自动编码。例如，

```
i=1.
while i<=CurrentSource.AutoCodeScripts.Count do
  CurrentSource.AutoCodeScripts.GetString(i,脚本名)
  writeln("调用自动脚本："+脚本名)
  i=i+1
endwhile.
```

3、段落

段落位于源文件之下，是源文件的属性之一，即 Source.Segment。它的关键词是Segment，具有 Segment.ID（段落身份号）、Segment.Codes（所有代码）、Segment.SuggestedCodes(所有推荐代码)、Segment.Memo(所有便笺)、Segment.CodesCount（代码总数）、Segment.CurrentSource（当前打开的源文件）、Segment.Scripts（所有脚本）、Segment.Links（所有链接）、Segment.NRS（段落描述，包括段落身份号、源文件名、段落在文件中的起点位置和终点位置）、Segment.MySource（源文件名称）等属性。

段落拥有自己专有的方法，包括：

（1）字符操作

- 查询字符：Segment.Contains("字符")，返回布尔值，如果含有该字符，返回值为TRUE，否则为FALSE。

（2）代码操作

- 检索代码：Segment.GetCode("韦伯",社会学家)，它有两个参数，一个是待检索的代码名，另一个是存储检索结果的列表变量，在这个示例中，前者是“韦伯”，后者是“社会学家”。返回布尔值，如果找到代码，返回值为TRUE，否则为FALSE。
- 增加代码：Segment.AddCode("韦伯")，给当前段落添加代码“韦伯”。它有一个参数，即代码名。没有返回值。
- 删除代码：Segment.RemoveCode("韦伯")，删除当前段落中的某个代码，在本例中为“韦伯”。它有一个参数，即代码名。没有返回值。
- 添加推荐代码:Segment.ApplySuggestions,给当前段落添加所有软件推荐的代码。它没有参数，没有返回值。
- 查询一个代码：Segment.HasCode("韦伯")，查询段落是否具有一个列表中的所有代码。返回布尔值，如果所有代码均在一个段落中，返回值为TRUE，否则为FALSE。
- 查询多个代码：Segment.HasAllCodes(List)，查询段落是否具有一个列表中的所有代码。返回布尔值,如果所有代码均在一个段落中,返回值为TRUE,否则为FALSE。
- 计算代码之间的相似度：Segment.AVSimilarityScore(List)，参数是包含代码的列表，返回值是一个整数，表示代码的相似度。
- 编码者：Segment.CodedBy("User")，用于查看代码的编码者，它可以有一个或两个参数。一个参数时，是编码者名。两个参数时，第一个字符串是代码名，第二个是编码者名。返回布尔值，如果该代码者编过代码，返回值为TRUE，否则为FALSE。
- 检索具有属性值的代码：Segment.GetCodeAssignment(代码变量,代码属性值变

量)，第一个参数是所要考察的代码，如果某个代码有属性值，它就会进入“代码属性值变量”中。

例如，我们可以用下列脚本来添加和删除代码：

{代码操作示例}

```
ForEach Segment in CurrentSource.Segments
    if (Segment.Contains("计划")) Then
      Segment.AddCode("主导型政府")
      Segment.ApplySuggestions
    endif
    if (Segment.GetCode("主导型政府",删除代码)) Then
      Segment.RemoveCode("主导型政府")
    Display(“我们已经在所有段落中删除了”+删除代码)
    endif
Endfor
```

4、代码

代码位于段落之中，是源文件的属性之一，即 Segment.Codes。它的关键词是 Code，具有 Code.Attributes（代码属性）、Code.Values（代码的属性值）、Code.Text（描述代码定义的文本）、Code.Memo（所有便笺）、Code.Links（代码链接）、Code.Synonyms（代码同义词）、Code.Plural（代码是否复数）和 Code.Article（是否有冠词）等属性。

代码拥有自己专有的方法，包括：

（1）查询代码的属性值

Code.GetAV("属性名",赋值列表变量)，它有两个参数，一个是属性名，另一个是赋值列表变量。返回布尔值，如果所有代码均在一个段落中，返回值为 TRUE，否则为 FALSE。例如，

```
If Project.GetCode("地方政府",代码) Then
  if (代码.GetAV("服务型",属性值)) Then
    writeln("属性值等于" + 属性值)
  EndIf
  else writeln("“地方政府”没有该属性值。")
EndIf.
```

（2）查询链接类型

Code.HasLink("关联")，它查询一个代码是否具有某种类型的链接，需要一个参数，即链接类型名。返回布尔值，如果代码具有这种类型的链接，返回值为 TRUE，否则为 FALSE。例如，

```
If Project.GetCode("地方政府",代码) Then
if (代码.HasLink("关联")) Then
  writeln("有" )
EndIf
else writeln("没有")
EndIf.
```

（3）查询代码之间的链接

Code.LinkedTo(方向,"链接类型","代码名")，它查询两个代码之间是否具有特定类型、特定方向的链接。它需要三个参数，第一个参数是方向，有三个关键词，即 FORWARD、BACKWARD 和 BOTH，分别表示正向、反向和双向)；第二个参数是链接类型；第三个是目标代码名。返回布尔值，如果两个代码之间具有该方向的链接，返回值为 TRUE，否则为 FALSE。例如，

```
if Project.GetCode("地方政府",代码) Then
if (代码.LinkedTo(BOTH,"关联","改制")) Then
  writeln("地方政府和改制之间有关联")
EndIf
else writeln("地方政府和改制之间没有关联")
EndIf
```

（4）编辑代码

Code.Edit，它启动代码编辑器，对当前代码“Code”进行编辑。没有返回值，也没有参数。例如，

```
if (Project.GetCode("地方政府", 代码)) then
代码.Edit
endif.
```

（5）代码的上下级代码

Code.GetAncestors()或 Code.GetDescendants()，它返回代码 Code 的所有上级代码或下级代码。两者均没有返回值，有一个参数，是接受返回值的一个列表变量。例如，

```
if Project.GetCode("地方政府",代码) Then
代码.GetAncestors(上级代码)
代码.GetDescendants(下级代码)
forEach item in 上级代码
  writeln(item)
endFor
forEach item in 上级代码
  writeln(item)
endFor
endIf.
```

（6）正向或反向链接的代码

Code.ForwardLinkages("链接",列表变量)或Code.BackwardLinkages("链接",列表变量)，它们返回和该代码具有正向链接或反向链接的代码列表，需要两个参数，第一个参数是链接名，第二个是接受返回值的列表变量。例如，

```
if Project.GetCode("地方政府",代码) Then
代码.ForwardLinkages("关联",正向关联的代码)
forEach item in 正向关联的代码
  writeln(item)
endFor
代码.BackwardLinkages("关联",正向关联的代码)
forEach item in 反向关联的代码
  writeln(item)
endFor
endIf.
```

5、链接

链接是代码的属性之一，即 Codes.Links，它的关键词是 Link，Link.Memo（所有便笺）、Link.Predictive（链接的预见性，取值为 PREDICTIVE 和 EXCLUSIVE）、Link.Symmetry（链

接的对称性，取值为 SYMETRIC 和 ASYMETRIC)、Link.Instances（该类链接的所有实例)、Link.FSS(链接的正向单数、单数动词短语)、Link.FSP(链接的正向单数、复数动词短语)、Link.FPS(链接的正向复数、单数动词短语)、Link.FPP(链接的正向复数、复数动词短语)、Link.BSS(链接的反向单数、单数动词短语)、Link.BSP(链接的反向单数、复数动词短语)、Link.BPS(链接的反向复数、单数动词短语)、Link.BPP(链接的反向复数、复数动词短语)、等属性。

链接拥有自己专有的方法，包括：

（1）检索实例

Link.FindInstancee('subjName','objName', linkInst1)，检索项目中是否具有特定主语和宾语的链接实例，需要三个参数，第一和第二个参数分别是主语和宾语，第三个参数是一个对象变量，存储检索所得到的结果。返回布尔值，如果主宾之间具有该类链接的实例，返回值为 TRUE，否则为 FALSE。

（2）检索参照

Link.GetReferences("代码",链接列表)、GetReferencedSubj("代码",链接列表)和 GetReferencedObj("代码",链接列表)是 Qualrus 中用于检索项目中的特定链接是否具有特定参照的三条指令。它们需要两个参数，第一个参数是代码名，第二个参数是一个列表变量，存储检索所得到的结果。返回布尔值，如果条件符合，返回值为 TRUE，否则为 FALSE。它们之间的差异是，GetReferences 不区分代码的主语或宾语属性，GetReferencedSubj 则要求代码处于主语位置，GetReferencedObj 要求代码处于宾语位置。例如，

```
if (Project.FindLink("因果", 因果链接)) then
  因果链接.GetReferences("创业",链接列表)
  因果链接.GetReferencesSubj("创业",主语链接列表)
  因果链接.GetReferencesObj("创业",宾语链接列表)
endif.
```

（3）检验参照

Link.HasReference（是否有参照)、Link.HasReferencedSubj（是否有参照主语)、Link.HasReferencedObj（是否有参照宾语)、Link.HasInstance（是否有实例)，这一组指令主要用于检索链接的参照。

当一个代码以某种类型的链接和其他代码关联时，我们称之为具有“参照”，此时，Link.HasReference("代码")返回 TRUE。在这种参照关系中，如果所考察的代码为主语，

则 Link.HasReferencedSubj("代码")返回 TRUE；如果所考察的代码为宾语，则 Link.HasReferencedObj("代码")返回 TRUE。这三个指令均需要一个参数，即代码名，返回布尔值，如果条件符合，返回值为 TRUE，否则为 FALSE。例如，

```
if (Project.FindLink("因果", aLink)) then
  if aLink.HasReference("创业") then
    writeln("创业和其他代码之间具有因果联系。")
  else writeln("创业和其他代码之间没有因果联系。")
  endif
  if aLink.HasReferencedSubj("创业") then
    writeln("创业代码以主语方式和其他代码之间具有因果联系。")
  else writeln("创业以主语方式和其他代码之间没有因果联系。")
  endif
  if aLink.HasReferencedObj("创业") then
    writeln("创业代码以宾语方式和其他代码之间具有因果联系。")
  else writeln("创业以宾语方式和其他代码之间没有因果联系。")
  endif
endif.
```

Link.HasInstance("代码甲","代码乙")和前面的指令稍有不同，它需要两个参数，分别是两个代码名，用来检查这两个代码之间是否具有特定链接类型。返回布尔值，如果条件符合，返回值为 TRUE，否则为 FALSE。例如，

```
if (Project.FindLink("因果", 因果链接)) then
  if 因果链接.HasReference("社会资本","创业") then
    writeln("社会资本和创业之间具有因果联系。")
  else writeln("社会资本和创业之间没有因果联系。")
  endif
endif.
```

（4）检索动词短语

Link.GetVerbPhrase("FSS")是 Qualrus 中用于检索项目中的特定链接属性中的动词短语，它需要一个参数，即动词短语的词性，取值为 FSS、FSP、FPS、FPP、BSS、BSP、BPS 或 BPP。返回布尔值，如果条件符合，返回值为 TRUE，否则为 FALSE。例如，

```
If Project.GetLink("因果",因果链接) then
   writeln(因果链接.GetVerbPhrase("FSS"))
else
  writeln("项目中没有因果链接！")
endIf.
```

6、列表

链接是项目中的对象，它相对游离，关键词是 List，其属性包括 List.Count（列表元素总数）、List.AsList（作为列表显示，作为 Display 的参数时，将显示全部列表元素）等。

列表拥有自己专有的方法，包括：

（1）列表操作

- 列表甲.Copyfrom(列表乙)，把列表乙的内容拷贝到列表甲。
- 列表甲.AND(列表乙)，合并列表乙中的元素到列表甲，不包括列表甲已有的元素。
- 列表甲.Intersects(列表乙),列表甲和列表乙的元素是否有交集。若有,返回 TRUE，没有则返回 FALSE。
- List.SaveToFile("文件名")，把 List 的内容存储到文件中，每一个元素占一行，文件存储在项目所默认的路径下。
- List.LoadFromFile("文件名")，把文件中的内容赋值给 List。

（2）对象操作

- List.AddObject 或 List.Add 用来给列表增加对象，两者的差异是，List.Add 只增加对象，List.AddObject 在增加对象的同时，还增加对象名。
- List.GetObject(i,对象)，把 List 中的第 i 个对象存储在“对象”变量中。

（3）元素操作

- List.Clear，将清空 List 中的元素，得到一个空集的列表。
- List.Delete(i)，将删除 List 中的第 i 个元素，注意，Qualrus 的元素指针从 1 开始。
- List.Member("字符",指针)，判定“字符”是否是 List 中的元素，若是，把其指针存储在“指针”列表中。若成功，返回值 TRUE，否则为 FALSE。
- List.Member("字符")，判定“字符”是否是 List 中的元素，若是返回值 TRUE，

否则为 FALSE。

- List.Sort，将 List 中的元素排序。

（4）字符操作

- List.GetString(i,字符表)，把 List 中的第 i 个字符存储到“字符表”中。
- List.InsertString(i,"String")，在 List 中的第 i 个位置上插入字符“String”。

例如，

```
i=1.
list(社会学家).{生成一个列表变量，名称为“社会学家”}
社会学家.AddString("韦伯").{在其中添加第一个元素，注意 Qualrus 列表元素的指针从 1 开始，AddString 是列表对象的一种方法}
社会学家.AddString("马克思").{在其中添加第二个元素}
社会学家.AddString("涂尔干").{在其中添加第三个元素}
社会学家.AddString("帕森斯").{在其中添加第四个元素}
display(社会学家.Count). {显示社会学家总数，结果是 4}
社会学家.Sort. {把社会学家中元素排序}
while (i<=社会学家.Count) do {Count 是列表的一种属性，即元素总数}
  社会学家.GetString(i,名单){把“社会学家”列表中的第 i 个元放入变量“名单”中，GetString 是列表对象的一种方法}
  writeln(名单) {现在的顺序是马克思、帕森斯、涂尔干、韦伯}
  i=i+1  {注意计数器的递增！}
endwhile.
社会学家.SaveToFile("社会学家.txt"). {存储在文件中}
社会学家.Clear. {清空社会学家中的内容}
社会学家.LoadFromFile("社会学家.txt"). {从文件中读取内容}
Display(社会学家.AsList).{以列表形式显示社会学家，结果是“马克思, 帕森斯, 涂尔干,and 韦伯”}
```

## （五）一个示例

下面，我们将用一个来演示用脚本做一个分割文本，并给每个段落添加一个作者代码的自动编码任务，我们将尽量用注释语句来解释每一个指令的作用。在使用下列脚本之前，

我们将文本的每个自然段落当作一个分析段落，并用[endPARA]作为段落分隔符[1]。

```
{标题，标题将增加输出结果的可读性}
writeln("<B>一个自动编码的脚本</B>").
writeln("-------------------------").
delimeter="[endPARA]". {界定段落分隔符}
author="Bourgois2001". {界定作者代码}
sourcetext=Project.CurrentSource.Text. {当前源文件的所有文本}
delimeterLength=length(delimeter). {段落分隔符的长度，9}
i=0.
delimeterCount=SubstringCount(delimeter,sourcetext). {计算分隔符在原文件中的个数}
list(codesList).  {界定一个代码列表变量，codesList}
number(previousstart). {界定一个数值变量，previousstart}
while i<=DelimeterCount do {循环次数由段落总数 DelimeterCount 决定}
  current=position(delimeter,sourcetext,i)+delimeterLength  {第 i 个段落分隔符的位置}
  if current=delimeterLength then current=0 endif {第一个段落，前面没有段落分隔符}
  next=position(delimeter,sourcetext,i+1) {第 i＋1 个段落分隔符的位置}
  if next=0
    then recordLength=length(sourcetext)-current+1 {最后一个段落的长度}
    else recordLength=next-current {段落的长度}
  endif
  if recordLength>0 then
    Project.FindCode(author, aCode) {项目是否有代码“author”，并复制到 aCode 变量中}
    codesList.AddObject(author, aCode) {把代码 “author”添加到 codeList 列表中}
    if Project.CurrentSource.CreateSegment(current,recordLength,codesList,aSegment) then
     {创建段落，第一个参数是起始位置，第二个参数是段落长度，第二个参数是代码列表，第四个参数是所生成的段落列表变量}
      aSegment.ApplySuggestions  {添加软件推荐的所有代码}
     endif
  endif
  i=i+1  {计数器递增}
endwhile.
Project.CurrentSource.Refresh.. {源文件刷新}
```

[1] 可以在 UltraEdit 中用“.[endPARA]^p”替代“.^p”完成这个操作。

在上面的示例中，我们把每一个自然段落都进行编码工作。一个更简明的做法是，如果定性分析辅助软件没有推荐代码，那么，就不必给这个自然段落编码。另外，上面的脚本需要对源文件的每一个自然段落事先插入一个分隔符，在下面的脚本中，我们将利用 Qualrus 的段落函数，简化这一工作。

```
writeln("<B>改进的自动编码脚本</B>").
writeln("<B>夏传玲，中国社会科学院社会学所</B>").
writeln("------要求项目中存在代码“Bruins1999”，否则出错！------").

list(codeList). {界定代码列表变量，codeList}
author="Bruins1999". {界定作者代码}

Project.GetCode(author, aCode).  {项目是否有代码“author”，并复制到 aCode 变量中}
codeList.AddObject("Bruins1999", aCode). {把代码“author”添加到 codeList 列表中}

if  CurrentSource.GetFirstParagraph(paraList)  {把当前源文件中的第一个自然段落赋值给变量，paraList}
then
    if MakeSegment(CurrentSource,paraList,codeList,segList) {从当前源文件的第 paraList 个自然段落，划定一个编码段落，赋值给变量 segList}
    then
        segList.ApplySuggestions {添加软件推荐的所有代码}
        if segList.CodesCount=1 {只有作者代码？}
        then CurrentSource.DeleteSegment(segList) {删除本编码段落}
        endif
    endif
endif.
while CurrentSource.GetNextParagraph(paraList) do {寻找下一个自然段落}
  if  MakeSegment(CurrentSource,paraList,codeList,segList) {从当前源文件的第 paraList 个自然段落，划定一个编码段落，赋值给变量 segList}
  then
    segList.ApplySuggestions {添加软件推荐的所有代码}
    if segList.CodesCount=1 {只有作者代码？}
```

```
      then CurrentSource.DeleteSegment(segList)   {删除本编码段落}
      endif
    endif
endwhile.
CurrentSource.Refresh. {源文件刷新}
List(AllCodes).   {定义列表}
AllCodes.COPYFROM(Project.Codes).   {赋值}
Display(AllCodes).   {显示}
AllCodes.SaveToFile(“AllCodes.txt”).   {所有代码存储在文件 AllCodes.txt 中}
writeln("-------完成编码任务----------").
```

# 七、附录

表 1 Qualrus 对象及其属性、方法一览表

| 对象 | 注释 |
|---|---|
| and | 和运算符 |
| AudioSegment.StartPos | 音频段落的起始位置 |
| AudioSegment.EndPos | 音频段落的中止位置 |
| AudioSegment.Len | 音频段落的长度 |
| AudioSource.MyLength | 音频源的长度 |
| Code.Name | 代码 Code 的名称 |
| Code.Links | 代码 Code 的所有链接 |
| Code.Plural | 代码 Code 的复数 |
| Code.Article | 代码 Code 的冠词 |
| Code.Synonyms | 代码 Code 的同义词 |
| Code.Attributes | 代码 Code 的默认属性 |
| Code.Attributes.Count | 代码 Code 的默认属性数 |
| Code.Attributes.GetString(i, attributeName) | 把代码 Code 的第 i 个属性名赋值给字符串变量 attributeName |
| Code.Values | 代码 Code 的默认属性取值 |
| Code.Text | 代码 Code 的定义部分 |
| Code.AVTemplate | 代码 Code 的属性模板 |
| Code.GetAncestors(ioList) | 上级代码，赋值给 ioList |
| Code.GetDescendants(ioList) | 下级代码，赋值给 ioList |
| Code.GetParents(ioList) | 父代码，赋值给 ioList |
| Code.GetChildren(ioList) | 子代码，赋值给 ioList |
| Code.Describe(ioList) | 把代码描述存入 ioList |
| Code.Describe(alist,ioList) | 按 aList 的顺序把把代码描述存入 ioList |
| Code.ISA(aCode) | Code 是 aCode 的一个实例吗？ |
| Code.ForwardLinkages(aLink,ioList) | 把 Code 的所有正向链接存入 ioList |
| Code.BackwardLinkages(aLink,ioList) | 把 Code 的所有反向链接存入 ioList |
| CodeAssignment.Code | 已编码的代码 |
| CodeAssignment.Why | 编码原因 |
| CodeAssignment.Attributes | 已赋值的属性 |
| CodeAssignment.Values | 已赋值属性值 |
| CodeAssignment.GetAV(attribute, VAR value) | 查询代码的属性及其值，成功则返回 TRUE。 |
| CodeAssignment.SetAV(attribute, value) | 设定代码的属性及其值，成功则返回 TRUE。 |
| CodeAssignment.FindWhy(WHY) | 查询编码原因，成功则返回 TRUE。 |
| CodeChange.Change | 代码变化 |
| CurrentSegment.Codes | 当前段落的所有代码 |
| CurrentSegment.HasCode(aCode) | 当前段落是否有代码 aCode，若有，返回 TRUE。 |
| CurrentSegment.SuggestAddCode(aCode. | 在当前段落，建议增加代码 aCode，原因是 aString。 |

| 对象 | 注释 |
| --- | --- |
| NAME,aString) | |
| CurrentSegment.SuggestDelCode(aCode. NAME,aString) | 在当前段落，建议删除代码 aCode，原因是 aString。 |
| CurrentSegment.SuggestedCodes | 当前段落所推荐的代码 |
| CurrentSegment.Text | 当前段落的文本内容 |
| CurrentSource | 当前打开的源文件 |
| CurrentSource.CreateSegment (current,recordLength,codesList,aSegment) | 在当前源文件中创建段落，第一个参数是起始位置，第二个参数是段落长度，第二个参数是代码列表，第四个参数是所生成的段落列表变量。 |
| CurrentSource.ISFIRST(aSegment) | 当前源文件中的 aSegment 是否为第一个段落，若是，返回 TRUE。 |
| CurrentSource.CurrentSegment | 当前源文件的当前段落 |
| CurrentSource.GetFirstLine(lineList) | 把当前源文件中的第一行文字存入 lineList 句子列表中。 |
| CurrentSource.GetNextline(lineList) | 把当前源文件中的下一行文字存入 lineList 句子列表中。 |
| CurrentSource.PrevSegment | 当前源文件当前段落的前一个段落 |
| display(变量名) | 显示变量内容 |
| forEach/in/endFor | 对列表中的所有元素元素进行操作，循环语句。 |
| if 条件 then 操作 endif | 条件语句 |
| Input(ArgsA)/Return(ArgsA) | 用于被调用的函数，接受传递过来的参数 ArgsA，返回参数 ArgsB。 |
| Link.Name | 链接名 |
| Link.Instances | 链接的实例数 |
| Link.Connectivity | 链接的指向属性 |
| Link.Symetry | 链接的对称属性，取值为“SYMETRIC” |
| Link.Transitive | 链接的传递属性，取值为“Transitive” |
| Link.Predictive | 链接的预测属性，取值为“Predictive”、“EXCLUSIVE” |
| Link.Definition | 链接的定义 |
| Link.FSS | 链接的正向单数、单数动词短语 |
| Link.FSP | 链接的正向单数、复数动词短语 |
| Link.FPS | 链接的正向复数、单数动词短语 |
| Link.FPP | 链接的正向复数、复数动词短语 |
| Link.BSS | 链接的反向单数、单数动词短语 |
| Link.BSP | 链接的反向单数、复数动词短语 |
| Link.BPS | 链接的反向复数、单数动词短语 |
| Link.BPP | 链接的反向复数、复数动词短语 |
| Link.GetReferences(aCode, ioList) | 和代码 aCode 具有 Link 链接的代码，添加到 ioList 中，其元素是该链接的实例。 |
| Link.GetReferencedSubj(aCode, ioList) | 链接实例中的主代码，添加到 ioList 中 |
| Link.GetReferencedObj(aCode, ioList) | 链接实例中的宾代码，添加到 ioList 中 |
| Link.GetVerbPhrase(["FSS","FSP",..."BPP"]) | 查询链接的动词 |
| LinkInstance.ID | 链接实例的身份号 |
| LinkInstance.MyLink | 查询链接实例的类型 |

| 对象 | 注释 |
| --- | --- |
| LinkInstance.Subj | 一个链接实例中的主代码，返回代码名 |
| LinkInstance.Obj | 一个链接实例中的宾代码，返回代码名 |
| LinkInstance.FVP | 链接实例的主动态 |
| LinkInstance.BVP | 链接实例的被动态 |
| LinkInstance.Name | 链接实例的名称 |
| LinkInstance.MakeSentence(方向) | 以方向（关键词为 FORWARD、BACKWARD）形成一个链接实例。 |
| list 元素 from 源头 to 列表变量 where 条件 endlist | 符合条件时，从源头把元素添加到列表变量中 |
| List.AddObject(aObject) | 把 aObject 添加到名为“List”的列表中 |
| List.Add(aString, aObject) | 把一个字符串 aString 添加到 aObject 中 |
| List.AddString(aString) | 在列表中添加字符串 aString |
| List.AND(bList,cList) | 合并 List 和 bList，结果是 cList |
| List.AsList | 以列表形式显示 List |
| List.Clear | 清空列表元素 |
| List.CopyFrom(bList) | 把 bList 中的列表拷贝到 List 中 |
| List.Count | 列表元素总数 |
| List.Delete(i) | 在列表中删除第 i 个元素 |
| List.GetObj(i, anObject) | 提取列表中第 i 个对象 |
| List.GetString(i, aString) | 提取列表中第 i 个字符串，到 aString 中。 |
| List.LoadFromFile(filename) | 从文件中载入列表 |
| List.Member(″string″) | 判定“字符”是否是 List 中的元素，若是返回值 TRUE，否则为 FALSE。 |
| List.Member(″string″, anObject) | 判定“字符”是否是 List 中的元素，若是返回值 TRUE，否则为 FALSE，并把它存储在一个对象列表中。 |
| List.Member(″string″, anIndex) | 判定“字符”是否是 List 中的元素，若是，把其指针存储在“指针”列表中。若成功，返回值 TRUE，否则为 FALSE。 |
| List.SaveToFile(filename) | 存储列表 |
| MakeSegment(CurrentSource,paraList,codeList,segList) | 从当前源文件的第 paraList 个自然段落，划定一个编码段落，赋值给变量 segList |
| not | 否运算符 |
| PictureSegment.Left | 图片段落的左坐标 |
| PictureSegmentTop | 图片段落的上坐标 |
| PictureSegment.Width | 图片段落的宽度 |
| PictureSegment.Height | 图片段落的高度度 |
| PictureSegment.MyRect | 图片段落的面积 |
| PictureSource.PictureWidth | 源图片的宽度 |
| PictureSource.PictureHeight | 源图片的高度 |
| Project.Name | 项目名 |
| Project.Path | 项目中的路径图 |
| Project.Sources | 项目的源文件 |
| Project.Codes | 项目中的所有代码列表 |
| Project.Links | 项目中的链接 |
| Project.Scripts | 项目中的脚本 |
| Project.Views | 项目中的示意图 |

| 对象 | 注释 |
| --- | --- |
| Project.Reports | 项目中的报告 |
| Project.Lists | 项目中的列表 |
| Project.CurrentUser | 项目的当前用户 |
| Project.AddList(objList) | 在项目中增加列表 objList |
| Project.DeleteSource(aFile) | 删除项目中的源文件 aFile |
| Project.DeleteCode(aCode) | 删除项目中的代码 aCode |
| Project.DeleteLink(aLink) | 删除项目中的链接 aLink |
| Project.DeleteScript(aScript) | 删除项目中的脚本 aScript |
| Project.DeleteView(aView) | 删除项目中的示意图 aView |
| Project.DeleteReport(aReport) | 删除项目中的报告 aReport |
| Project.DeleteList(aList) | 删除项目中的列表 aList |
| Project.GetSource(aSource, objList) | 载入源文件 aSource，放入对象 objList 列表中，若成功，返回 TRUE。 |
| Project.GetCode(aCode,codeList) | 从项目中提取 aCode 代码，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.GetLink(aLink,objList) | 从项目中提取 aLink 列表，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.GetScript(aScript,objList) | 从项目中提取 aScript 脚本，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.GetView(aView,objList) | 从项目中提取 aView 示意图，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.GetReport(aReport, objList) | 从项目中提取 aReport 报告，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.GetList(aList,objList) | 从项目中提取 aList 列表，放入 objList 中，若成功，返回 TRUE，否则返回 FALSE。 |
| Project.Backup | 备份整个项目 |
| Project.Details | 弹出项目细节窗口 |
| Project.Describe(ioList) | 项目描述存入 ioList 中。 |
| Report.Name | 报告名 |
| Report.ReportFile | 报告文件 |
| Report.Items | 报告中的元素 |
| Script("CalledScript",ArgsA,ArgsB) | 调用一个脚本，名为 CalledScript，传递的值位于第一个参数 ArgsA 中，返回的结果存储在第二个参数 ArgsB 中 |
| Script.Name | 脚本名 |
| Script.AutoCodeable | 是否可自动运行 |
| Segment(segList) | 界定一个段落变量，segList |
| Segment.ID | 段落的身份号 |
| Segment.MySource | 段落的源文件名称 |
| Segment.Codes | 段落中的代码 |
| Segment.SuggestedCodes | 段落中软件推荐的代码 |
| Segment.NRS | 段落描述 |
| Segment.AddCode(aCode, WHY) | 给段落添加代码 aCode 及其原因。 |
| Segment.DelCode(aCode) | 删除已编的代码 aCode |
| Segment.HasCode(aCode) | 查询段落是否有代码 aCode，若有返回 TRUE |

| 对象 | 注释 |
| --- | --- |
| Segment.HasAnyCodes(aCodes) | 查询段落是否有 aCode 代码中的任何一个,若有返回 TRUE |
| Segment.HasAllCodes(aCodes) | 查询段落是否有 aCode 代码中的全部，若有返回 TRUE |
| Segment.GetCode(aCode, ioList) | 查询段落编码中是否有 aCode，结果入 ioList。 |
| Segment.GetCodeAssignment(aCode, ioList) | 查询段落编码中是否有 aCode 的赋值，结果入 ioList。 |
| Segment.SuggestCode(aCode, why) | 推荐代码 aCode 及其原因。 |
| Segment.SuggestedAddCode(aCode,why) | 添加推荐代码 aCode 及其原因。 |
| Segment.SuggestDelCode(aCode,why) | 删除推荐代码 aCode 及其原因。 |
| Segment.HasSuggestedCode(aCode) | 段落的推荐代码中是否有 aCode。 |
| Segment.Details | 段落细节 |
| Segment.CodedBy(aUser) | 当前代码由 aUser 编码，若成功，返回 TRUE。 |
| Segment.SimCodes(CurrentSegment.Codes,0.5,FALSE,ScoresList,SimSegsList) | 代码相似度函数，对当前段落中的代码进行相似度判断。需要五个参数，第一个是当前段落的代码，第二个参数是相似度阈限，当相似分数高于该阈限时，就会返回段落，第三个参数是是否排除完全相同的段落（取值为 TRUE 或 FALSE),第四和第五个参数分别是纪录结果的相似度分值和相似段落的列表。 |
| Segment.SimWords(CurrentSegment.Text,0.75,FALSE,ScoresList,SimSegsList) | 文本相似度函数，对当前段落中的文本进行相似度判断。需要五个参数，第一个是当前段落的文本，第二个参数是相似度阈限，当相似分数高于该阈限时，就会返回段落，第三个参数是是否排除完全相同的段落（取值为 TRUE 或 FALSE),第四和第五个参数分别是纪录结果的相似度分值和相似段落的列表。 |
| Source.Name | 源文件名 |
| Source.Segments | 源文件的段落 |
| Source.SegmentsWithin(aSegment,PROJECT,SegList) | 判断在段落 aSegment 是否在 Project、Source 等范围中。若是，把 aSegment 放入段落变量 SegList 中。 |
| Source.CurrentSegment | 源文件的当前段落 |
| Source.SegmentByID(inID) | 段落的身份号 |
| Source.PrevSegment | 当前段落的前一个段落 |
| Source.IsFirst(aSegment) | aSegment 是当前文件中的第一个段落吗？若是，返回 TRUE。 |
| Source.SeekSegment(方向, aScript, aSegment) | 按“方向”（关键词为 FORWARD、BACKWARD）检索段落，并执行脚本，生成的段落存入 aSegment。 |
| Source.SeekSegments(方向, aScript, ioList) | 按“方向”（关键词为 FORWARD、BACKWARD）检索段落，并执行脚本，生成的段落存入 aSegment。 |
| Split("字符串"或字符变量, "分隔符", 结果列表) | 将字符串或字符变量中的字符，用分隔符进行分割。 |
| TextSegment.StartPos | 文本段落的起始位置 |
| TextSegment.Len | 文本段落的长度 |
| TextSegment.Text | 文本段落的全文 |
| TextSegment.Contains(aText) | 文本段落是否有 aText，若有，返回 TRUE。 |
| TextSegment.ContainsAny(aString) | 段落中是否含有 aString 字符变量的任一字符，若是，返回 TRUE。 |
| TextSegment.ContainsAll(aString) | 段落中是否含有 aString 字符变量的所有字符，若是，返 |

| 对象 | 注释 |
| --- | --- |
| | 回 TRUE。 |
| TextSource.Contains(aText) | 文本源文件是否有 aText，若有，返回 TRUE。 |
| VideoSegment.StartPos | 视频段落的起始位置 |
| VideoSegment.EndPos | 视频段落的中止位置 |
| VideoSegment.Len | 视频段落的长度 |
| VideoSource.MyLength | 视频源文件的总长度 |
| View.Name | 示意图名 |
| View.RunMode | 示意图运行方式 |
| View.ShortLabels | 示意图短标签 |
| View.BackColor | 示意图背景色 |

参考文献

Blanco, Cheng. 2003. "Qualrus." *Social Science Computer Review* 21(2):257-260.

Brent, Edward, Pawel Slusarz, & Alan Thompson. 2002. *Qualrus Manual*. Columbia, MO: Idea Works, Inc.

Glaser, Barney S. & Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Publishing Company.

Kaplan, Abraham. 1964. *The Conduct of Inquiry: Methodology for Behavioral Science*. San Francisco,: Chandler Pub. Co.

Leake, David B. 1996. *Case-Based Reasoning: Experiences, Lessons & Future Directions*. Cambridge, Mass.: MIT Press.

Mill, John Stuart. 1872. *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence and the Methods of Scientific Investigation*. New York,: Harper & brothers.

Ragin, Charles C. 1987. *The Comparative Method: Moving Beyond Qualitative and Quantitative Strategies*. Berkeley: University of California Press.

Strauss, Anselm L. & Juliet M. Corbin. 1998. *Basics of QualItative Research : Techniques and Procedures for Developing Grounded Theory*, 2nd. Thousand Oaks: Sage Publications.

格尔兹. 1999. *文化的解释*. 纳日碧力戈等译. 上海: 上海人民出版社.