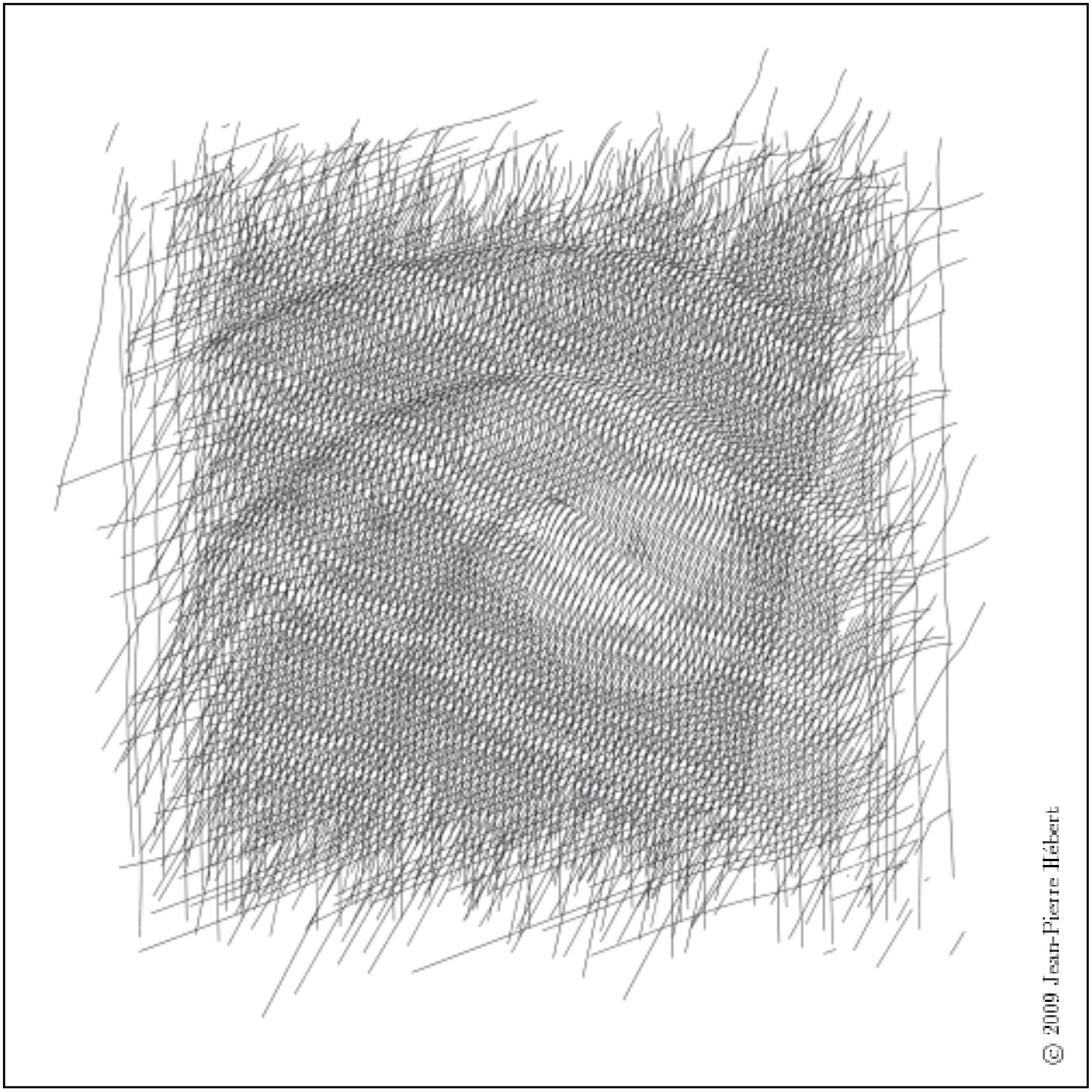

第 9 章。记录

本章介绍程序员可以定义新数据类型或记录类型的方法，每种类型都与所有其他类型不同。记录类型确定该类型的每个实例具有的字段的数量和名称。记录是通过定义记录类型窗体或生成记录类型描述符过程定义的。

第 9.1 节. 定义记录

定义记录类型窗体定义记录类型，并随之定义该类型记录的构造函数过程、仅对该类型的记录返回 true 的类型谓词、每个字段的访问过程以及每个可变字段的分配过程。例如，定义

```
（定义记录类型点（字段 x y））
```

创建具有两个字段 x 和 y 的点记录类型，并定义以下过程：

```
（制造点 x y）  构造 函数
（点? obj）     谓语
（点-x p）      字段 x 的访问器
（点-y p）      字段 y 的访问器
```

有了这个定义，我们可以使用这些过程来创建和操作点类型的记录，如下图所示。

```
（定义 p （make-point 36 -17））
（点? p） ⇒ #t
（点?’（缺点 36 -17）） ⇒ #f
（点-x p） ⇒ 36
（点-y p） ⇒ -17
```

默认情况下，字段是不可变的，但可以声明为可变字段。在下面点的替代定义中，x 字段是可变的，而 y 仍然是不可变的。

```
（定义记录类型点（字段（可变 x）y））
```

在这种情况下，除了上面显示的其他产品之外，define-record-type 还为 x 字段定义了一个突变体。

（点 x 设置！p x）字段 x 的突变体

变量可用于更改 x 字段的内容。

```
（定义 p （make-point 36 -17））
（point-x-set！ p （- （point-x p） 12））
（point-x p） ⇒ 24
```

为了清楚起见，字段可以显式声明为不可变;下面点的定义等同于上面的第二个定义。

```
（定义记录类型点（字段 （可变 x） （不可变 y））））
```

默认情况下，由 define-record-type 定义的过程的名称遵循上述示例所示的常规命名约定，但如果需要，程序员可以重写默认值。根据以下点定义，构造函数是 mkpoint，谓词是 ispoint？，x 和 y 的访问器是 x-val 和 y-val。x 的突变体是 set-x-val！。

```
（定义记录类型（点 mkpoint ispoint？）
（字段 （可变 x x-val set-x-val！
（不可变 y y-val））））
```

默认情况下，记录定义在每次计算时都会创建一个新类型，如下面的示例所示。

```
（define （f p）
（define-record-type point （fields x y））
（if （eq？ p ’make） （make-point 3 4） （point？
p））））
（f （f ’make））  ⇒ #f
```

对 f 的第一个（内部）调用返回一个点 p，该点 p 在第二个（外部）调用中传递给 f，该调用将点应用于 p。此点？ 是查找由第二次调用创建的类型的点，而 p 是第一次调用创建的类型的点。所以点？返回#f。

可以通过在记录定义中包含非生成子句来覆盖此默认生成行为。

```
（define （f p）
（define-record-type point （fields x y）
（nongenerative））
（if （eq？ p ’make） （make-point 3 4） （point？
p））））
（定义 p （f ’make））
（f p） ⇒ #t
```

以这种方式创建的记录类型仍与由出现在程序不同部分中的定义创建的记录类型不同，即使这些定义在语法上是相同的：

```
（定义 （f）
（定义记录类型点 （字段 x y） （非生成））
（使点 3 4））
（定义 （g p）
（定义记录类型点 （字段 x y） （非生成））
（点？ p））
（g （f）） ⇒ #f
```

即使这样也可以通过在非生成子句中包含 uid（唯一 id）来覆盖：

```
（定义 （f）
（定义记录类型点 （字段 x y）
（非生成性真正相同的点））
（使点 3 4））
（定义 （g p）
```

```
（定义记录类型点 （字段 x y）
（非生成性真正相同的点））
（点？ p））
（g （f）） ⇒ #t
```

uid 可以是任何标识符，但鼓励程序员从 RFC 4122 UUID 命名空间 [20] 中选择 uid，可能以记录类型名称作为前缀。

可以将记录类型定义为现有“父”类型的子类型，其父子句声明现有记录类型的名称。如果指定了父级，则新的“子”记录类型将继承父记录的字段，并且子类型的每个实例都被视为父类型的实例，以便可以在子类型的实例上使用父类型的访问器和变种函数。

```
（定义记录类型点（字段 x y））
（定义记录类型的 cpoint（父点）（字段颜色））
```

子类型具有父类型的所有字段，以及在子类型定义中声明的其他字段。这反映在 cpoint 的构造函数中，它现在采用三个参数，父参数后跟子参数。

```
（定义 cp （make-cpoint 3 4 'red））
```

子类型的记录被视为父类型的记录，但父类型的记录不是新类型的记录。

```
（点？ （make-cpoint 3 4 'red））⇒ #t
（cpoint？ （要点3 4））⇒ #f
```

仅为 cpoint 创建了一个新访问器，一个用于新字段颜色的访问器。父类型的现有访问器和变种器可用于访问和修改子类型的父字段。

```
（定义 cp （make-cpoint 3 4 'red））
（点 x cp） ⇒ 3
（点 y cp） ⇒ 4
（点-彩色 cp） ⇒ 红色
```

正如到目前为止给出的示例所示，由 define-record-type 定义的默认构造函数接受的参数与记录具有的字段（包括父字段和父字段的父字段等）一样多。程序员可以重写默认值，并为新类型的构造函数指定参数，以及如何通过协议子句确定构造记录字段的初始值。以下定义创建具有三个字段的点记录：x、y 和 d，其中 d 表示从原点的位移。构造函数仍然只接受两个参数，x 和 y 值，并将 d 初始化为 x 和 y 的平方和的平方根。

```
(define-record-type point
(fields x y d)
(protocol
(lambda （new）
(lambda （x y）
(new x y （sqrt （+ （* x x） （* y y））））

（定义 p （make-point 3 4））
(point-x p) ⇒ 3
(point-y p) ⇒ 4
(point-d p) ⇒ 5
```

协议子句中表达式的过程值接收原始构造函数 new 作为参数，并返回最终构造函数 c。基本上对 c 允许执行的操作没有限制，但是如果它返回，它应该返回调用 new 的结果。在此之前，它可能会修改新的记录实例（如果记录类型具有可变字段）、向某些外部处理程序注册它、打印消息等。在这种情况下，c 接受两个参数 x 和 y，并将 new 应用于 x、y 以及基于 x 和 y 计算原点位移的结果。

如果指定了父记录，则构造协议将变得更加复杂。cpoint 的以下定义假定该点已按上图所示进行定义。

```
(define-record-type cpoint
(parent point)
(fields color)
(protocol
(lambda (pargs->new)
(lambda (c x y)
( (pargs->new x y) c) ) ) )

(定义 cp (make-cpoint 'red 3 4) )
(point-x cp) ⇒ 3
(point-y cp) ⇒ 4
(point-d cp) ⇒ 5
(cpoint-color cp) ⇒ red
```

由于存在父子句，因此协议子句中表达式的过程值将接收一个过程 pargs->new，当应用于父参数时，将返回一个新过程。当传递子字段的值时，新过程将返回将父协议应用于其自己的相应新过程的结果。在这种情况下，pargs->new 传递子构造函数的第二个和第三个参数的值（x 和 y 值），并且生成的新过程传递子构造函数的第一个参数的值（颜色）。因此，此示例中提供的协议有效地反转了参数的正常顺序，其中父参数位于子参数之前，同时安排传递父协议所需的参数。

默认协议等效于

```
(lambda (新) 新)
```

对于没有父级的记录类型，而对于具有父级的记录类型，默认协议等效于以下内容

```
(lambda (pargs->new)
(lambda (x1 ... xn y1 ... ym)
((pargs->new x1 ... xn) y1 ... ym)))
```

其中 n 是父字段（包括祖父级等）字段的数量，m 是子字段的数量。

使用协议子句可使子记录定义与父记录类型的某些更改隔离开来。父定义可以修改为添加或删除字段，甚至添加、删除或更改父级，但只要父协议不更改，子协议和构造函数就不需要更改。

定义记录类型的其他详细信息和选项在下面的正式描述中给出。

语法： (定义记录类型记录名子句 ...)
语法： (定义记录类型 (记录名称构造函数 pred) 子句 ...)
库： (rnrs 记录语法)， (rnrs)

定义记录类型窗体或记录定义是一个定义，可以出现在其他定义可能出现的任何位置。它定义由记录名称标识的记录类型，以及记录类型的谓词、构造函数、访问器和变种函数。如果记录定义采用上面的第一种形式，则构造函数和谓词的名称派生自记录名称：构造函数的 make-record-name 和谓词的记录名称？。如果记录定义采用上面的第二种形式，则构造函数的名称是构造函数，谓词的名称是 pred。记录定义定义定义的所有名称都限定在记录定义出现的位置。

记录定义的子句...确定记录类型的字段及其访问器和变种符的名称;其父类型（如果有）;其施工协议;它是否是

非生成的，如果是，是否指定了其 uid;是否密封;以及它是否不透明。下面介绍了每个子句的语法和影响。

这些条款都不是必需的;因此，最简单的记录定义是

（定义记录类型记录名称）

它定义了一个新的、生成的、非密封的、非不透明的记录类型，没有父级和字段，以及一个没有参数和谓词的构造函数。

每种类型的子句中最多可以存在一种子句，如果存在父子句，则父-rtd 子句不得存在。出现的子句可以按任何顺序出现。

字段子句。（字段字段规范 ...） 子句声明记录类型的字段。每个现场规范必须采用以下形式之一：

字段名
（immmutable field-name）
（可变字段名）
（immmutable field-name accessor-name）
（可变字段名访问器-name mutator-name）

其中字段名称、访问器名称和突变名称是标识符。第一种形式，字段名称，等效于（不可变字段名称）。声明为不可变的字段的值不能更改，也不会为其创建任何突变体。对于前三种形式，访问器的名称是 rname-fname，其中 rname 是记录名称，fname 是字段名称。对于第三种形式，访问器的名称是 rname-fname-set！。第四和第五种形式显式声明访问器和突变体名称。

如果不存在字段子句或列表字段规范...为空，则记录类型没有字段（父字段除外，如果有）。

父子句。（父父名称）子句声明父记录类型;父名必须是先前通过定义记录类型定义的非密封记录类型的名称。记录类型的实例也被视为其父记录类型的实例，除了通过字段子句声明的字段外，还具有其父记录类型的所有字段。

非生成性子句。非生成性子句可以采用以下两种形式之一：

（非遗传性）
（非生成性 uid）

其中 uid 是一个符号。第一种形式等效于第二种形式，具有由实现在宏扩展时生成的 uid。计算具有非生成性子句的定义记录类型窗体时，当且仅当 uid 不是现有记录类型的 uid 时，才会创建新类型。

如果它是现有记录类型的 UID，则父记录、字段名称、密封属性和不透明属性必须匹配，如下所示。

- 如果指定了父记录类型，则现有记录类型必须具有相同的父 rtd（按 eqv？）。如果未指定父记录，则现有记录类型不得具有父记录类型。
- 必须提供相同数量的字段，具有相同的名称和相同的顺序，并且每个字段的可变性必须相同。
- 如果存在（密封#t）子句，则必须密封现有记录类型。否则，不得密封现有记录类型。
- 如果存在（不透明#t）子句，则现有记录类型必须不透明。否则，当且仅当指定了不透明的父类型时，现有记录类型必须不透明。

如果满足这些约束，则不会创建新的记录类型，并且记录类型定义的其他产品（构造函数、谓词、访问器和赋值函数）将对现有类型的记录进行操作。如果不满足这些约束，则实现可能会将其视为语法冲突，或者可能会引发具有条件类型 &断言的运行时异常。

对于非生成子句的第一种形式，仅当多次执行相同的定义时，生成的 uid 才能是现有记录类型的 uid，例如，如果它出现在多次调用的过程的正文中。

如果 uid 不是现有记录类型的 uid，或者不存在非生成子句，则会创建新的记录类型。

协议条款。（协议表达式）确定生成的构造函数用于构造记录类型的实例的协议。它必须评估为一个过程，并且此过程应是记录类型的适当协议，如第 326 页所述。

密封条款。表单的密封子句（密封#t）声明记录类型已密封。这意味着它不能被扩展，即不能用作另一个记录定义或制作记录类型描述符调用的父级。如果不存在密封子句，或者存在其中一种形式（密封#f），则记录类型未密封。

不透明子句。表单的不透明子句（不透明#t）声明记录类型是不透明的。不透明记录类型的实例不被视为记录？谓词或更重要的是，rtd 提取过程 record-rtd 的记录，这两者都在第 9.3 节中进行了描述。因此，无权访问记录名、访问器或变种人的代码无法访问或修改不透明记录类型的任何字段。如果记录类型的父级不透明，则该记录类型也是不透明的。如果不存在不透明子句，或者如果存在其中一种形式（不透明#f），并且父级（如果有）不透明，则记录类型不透明。

父 rtd 子句。（parent-rtd parent-rtd parent-rcd） 子句是父子句的替代项，用于指定父记录类型以及父记录构造函数描述符。当父 rtd 和 rcd 是通过调用 make-record-type-描述符和 make-record-constructor-描述符而获得的时，它主要有用。

父 rtd 的评估结果必须为 rtd 或#f。如果父 rtd 的计算结果为#f，则父 rcd 也必须计算#f。否则，父 rcd 的计算结果必须为 rcd 或#f。如果 parent-rcd 的计算结果为 rcd，它必须将 rtd 等效项（按 eqv？）封装到 parent-rtd 的值。如果 parent-rcd 的值#f，则它被视为具有默认协议的 parent-rtd 值的 rcd。

定义记录类型窗体的设计方式是，编译器通常可以确定它定义的记录类型的形状，包括所有字段的偏移量。但是，当使用 parent-rtd 子句时，此保证不成立，因为在运行时之前可能无法确定父 rtd。因此，只要父子句足够，父子句就优先于父子句。

语法： 字段
语法： 可变
语法： 不可变
语法： 父
语法： 协议
语法： 密封
语法： 不透明
语法： 非生成语法
：父 rtd
库：（rnrs 记录语法）、（rnrs）

这些标识符是定义记录类型的辅助关键字。引用这些标识符是语法冲突，除非在上下文中将它们识别为辅助关键

字。

第 9.2 节. 程序接口

过程（创建记录类型描述符）接口也可用于创建新的记录类型。过程接口比句法接口更灵活，但这种灵活性会导致程序的可读性和效率降低，因此程序员应该在足够的时候使用句法接口。

过程：（make-record-type-描述符名称 parent uid s? o? fields)
返回：新的或现有的记录类型库的记录类型
描述符 （rtd）：（rnrs 记录过程）、（rnrs）

name 必须是符号，父项必须是#f或非密封记录类型的 rtd，uid 必须是#f或符号，并且字段必须是矢量，其中每个元素都是表单（可变字段名称）或（不可变字段名称）的双元素列表。字段名称 字段名称 字段名称 ... 必须是符号，并且不必彼此不同。

如果 uid #f或不是现有记录类型的 uid，则此过程将创建新的记录类型，并为新类型返回记录类型描述符（rtd）。该类型具有父类型（第325页）由父描述，如果不是false;由 uid 指定的 uid，如果非 false;和字段指定的字段。它是密封的（第330页），如果s? 是非 false。如果不透明是不透明的或父级（如果指定）是不透明的，则它是不透明的（第 330 页）。新记录类型的名称是 name，字段的名称是字段名称 ...。

如果 uid 是非 false 并且是现有记录类型的 uid（第 325 页），则父级、字段、s? 和 o? 参数必须与现有记录类型的相应特征匹配。也就是说，父项必须相同（通过

eqv? 字段的数量必须相同;字段必须具有相同的名称，顺序相同，并且具有相同的可变性;s? 当且仅当现有记录类型已密封时，必须为 false;并且，如果未指定父项或不透明，则当且仅当现有记录类型不透明时，o? 必须为 false。如果是这种情况，则生成记录类型描述符将返回现有记录类型的 rtd。否则，将引发条件类型 &断言的异常。

使用 make-record-type-描述符返回的 rtd，程序可以动态生成构造函数、类型谓词、字段访问器和字段变量。下面的代码演示如何使用过程接口创建点记录类型和关联定义，类似于第 9.1 节中的第二个点记录定义，具有可变的 x 字段和不可变的 y 字段。

```
（定义 point-rtd （make-record-type-descriptor ’point
#f #f #f #f
’#（mutable x） （immutable y））））
（define point-rcd （make-record-constructor-descriptor
point-rtd
#f #f））
（define make-point （record-constructor point-rcd））
（define point? （记录谓词点 rtd））
（定义 point-x （record-accessor point-rtd 0））
（define point-y （record-accessor point-rtd 1））
（定义 point-x-set! （记录突变点 rtd 0））
```

请参阅本节末尾给出的其他示例。

过程：（记录类型描述符？obj）
返回：如果 obj 是记录类型描述符，则#f，否则#f
库：（rnrs 记录过程）、（rnrs）

请参阅本节末尾给出的示例。

过程：（make-record-constructor-descriptor rtd parent-rcd protocol）
返回：a record-constructor descriptor（rcd）
libraries：（rnrs records procedure），（rnrs）

仅 rtd 就足以创建谓词、访问器和赋值函数。但是，若要创建构造函数，首先需要为记录类型创建记录构造函数描述符（rcd）。rcd 封装了三条信息：已为其创建 rcd 的记录类型的 rtd、父 rcd（如果有）和协议。

父 rcd 参数必须是 rcd 或#f。如果是 rcd，则 rtd 必须具有父 rtd，并且父 rtd 必须与封装在父 rtd 中的 rtd 相同。如果 parent-rcd 为假，则 rtd 没有父级，或者假定父级具有默认协议的 rcd。

协议参数必须是过程或#f。如果#f，则假定使用默认协议。协议在第 326 页讨论。

请参阅本节末尾给出的示例。

语法：（记录类型描述符记录名）
返回：由记录名
语法标识的记录类型的 rtd：（记录构造函数描述符记录名）
返回：由记录名
库标识的记录类型的 rcd：（rnrs 记录语法）、（rnrs）

每个记录定义在后台为定义的记录类型创建 rtd 和 rcd。这些程序允许像任何其他rtd或rcd一样获得和使用 rtd和rcd。记录名必须是以前通过定义记录类型定义的记录的名称。

过程：（记录构造函数 rcd）
返回：封装在 rcd
库中的记录类型的记录构造函数：（rnrs 记录过程）、（rnrs）

记录构造函数的行为由协议确定，父 rcd（如果有）也封装在 rcd 中。

请参阅本节末尾给出的示例。

过程：（记录谓词 rtd）
返回：rtd
库的谓词：（rnrs 记录过程）、（rnrs）

此过程返回一个谓词，该谓词接受一个参数，如果该参数是 rtd 描述的记录类型的实例，则返回#t，否则#f。

请参阅本节末尾给出的示例。

过程：（记录访问器 rtd idx）
返回：idx
库指定的 rtd 字段的访问器：（rnrs 记录过程）、（rnrs）

idx 必须是小于 rtd 字段数的非负整数，不计算父字段。idx 值 0 指定在创建记录类型的定义记录类型窗体或 make-record-type-描述符调用中给出的第一个字段，1 指定第二个字段，依此类推。

子 rtd 不能直接用于为父字段创建访问器。若要为父字段创建访问器，必须改用父字段的记录类型描述符。

请参阅本节末尾给出的示例。

过程：（记录突变体 rtd idx）
返回：idx
库指定的 rtd 字段的突变体：（rnrs 记录过程）、（rnrs）

idx 必须是小于 rtd 字段数的非负整数，不计算父字段。idx 值 0 指定在创建记录类型的定义记录类型窗体或 make-record-type-描述符调用中给出的第一个字段，1 指定第二个字段，依此类推。指示的字段必须是可变的;否则，将引发条件类型 &断言的异常。

子 rtd 不能直接用于为父字段创建突变体。若要为父字段创建突变体，必须改用父字段的记录类型描述符。

下面的示例演示如何使用本节中描述的过程创建父记录类型和子记录类型、谓词、访问器、赋值函数和构造函数。

```
（定义 rtd/parent
 （make-record-type-descriptor ’parent #f #f #f #f
’#（mutable x））））

（记录类型描述符？ rtd/parent） ⇒ #t
（定义 parent？（记录谓词 rtd/父项））
（定义 parent-x （record-accessor rtd/parent 0））
（定义 set-parent-x！（记录突变体 rtd/父级 0））

（定义 rtd/child
 （make-record-type-descriptor ’child rtd/parent #f #f
#f
’#（mutable x） （immutable y））））

（define child？（记录谓词 rtd/子项））
（定义子 x （记录访问器 rtd/子 0））
（定义 set-child-x！（记录突变体 rtd/子 0））
（定义子级（记录访问器 rtd/子级 1））
```

```
(记录突变体 rtd/子项 1) ⇒ 异常：不可变字段（定义 rcd/父级

(make-record-constructor-描述符 rtd/parent #f
(lambda (new) (lambda (x) (new (* x x)))))

)(记录类型描述符? rcd/parent) ⇒ #f

(定义生成父级（记录构造函数 rcd/父级））

(定义 p (make-parent 10))
(parent? p) ⇒ #t
(parent-x p) ⇒ 100
(set-parent-x! p 150)
(parent-x p) ⇒ 150

(define rcd/child
(make-record-constructor-descriptor rtd/child rcd/parent
(lambda (pargs->new)
(lambda (x y)
((pargs->new x) (+ x 5) y)))

(定义生成子项（记录构造函数 rcd/子项））
(定义 c (make-child 10 'cc))
(parent? c) ⇒ #t
(child? c) ⇒ #t
(child? p) ⇒ #f

(parent-x c) ⇒ 100
(child-x c) ⇒ 15
(child-y c) ⇒ cc

(child-x p) ⇒ 异常: 无效的参数类型
```

第 9.3 节.检查

本节介绍用于询问有关记录类型描述符 （rtds） 的问题或从中提取信息的各种过程。它还描述了 record-rtd 过程，通过该过程可以提取非不透明记录实例的 rtd，从而允许检查实例的记录类型，并通过从 rtd 生成的记录访问器和变种器检查或修改记录本身。这是一项强大的功能，允许对便携式记录打印机和检查器进行编码。

无法从不透明记录类型的实例中提取记录类型描述符。这是区分不透明和非不透明记录类型的功能。

过程： （记录类型名称 rtd）
返回： 与 rtd
库关联的名称： （rnrs 记录检查）、 （rnrs）

```
（定义 record->name
（lambda （x）
（and （record? x） （record-type-name （record-rtd
x） ） ） ） ）
```

```
（定义记录类型暗淡（字段 w l h））
（记录>名称 （make-dim 10 15 6）） ⇒ dim
```

```
（定义记录类型 dim （字段 w l h） （不透明#t））
（记录>名称 （make-dim 10 15 6）） ⇒ #f
```

过程： （记录类型父 rtd）
返回： rtd 的父级，或者如果它没有父
库，则返回#f： （rnrs 记录检查）、 （rnrs）

```
（定义记录类型点（字段 x y））
（define-record-type cpoint （parent point） （fields
color））
（record-type-parent （record-type-descriptor point））
```

```
⇒ #f
（record-type-parent （record-type-descriptor
cpoint））⇒ #<rtd>
```

过程：（记录类型 uid rtd）
返回：rtd 的 uid，或者如果它没有 uid
库，则#f：（rnrs 记录检查）、（rnrs）

在没有程序员提供的 uid 的情况下创建的记录类型是否实际上具有一个记录类型，这完全取决于实现，因此此过程永远不能保证返回#f。

```
（定义记录类型点（字段 x y））
（定义记录类型 cpoint
（父点）
（字段颜色）
（非生成性 e40cc926-8cf4-4559-a47c-cac636630314））
（记录类型 uid （记录类型描述符点））⇒ 未指定
（记录类型 uid （记录类型描述符 cpoint））⇒
e40cc926-8cf4-4559-a47c-cac636630314
```

过程：（记录类型生成？rtd）
返回：#t rtd描述的记录类型是否为生成式，#f否则
过程：（记录类型密封？rtd）
返回：#t rtd描述的记录类型是否密封，#f否则
过程：（记录类型不透明？rtd）
返回：#t rtd描述的记录类型是否不透明，#f否则
库：（rnrs记录检查），（rnrs）

```
（定义记录类型表
（字段键值）
（不透明#t））
（定义 rtd （记录类型描述符表））
（记录类型生成？ rtd）⇒ #t
（记录类型密封？ rtd）⇒ #f
（记录类型不透明？ rtd）⇒ #t
```

```
（定义记录类型缓存表
（父表）
（字段键值）
（非生成性））
（定义 rtd （记录类型描述符缓存表））
（记录类型生成性？ rtd） ⇒ #f
（记录类型密封？ rtd） ⇒ #f
（记录类型不透明？ rtd） ⇒ #t
```

过程：（记录类型字段名称 rtd）
返回：一个向量，其中包含 rtd 库描述的类型的字段的名称：（rnrs 记录检查）、（rnrs）

此过程返回的向量是不可变的：修改它对 rtd 的影响是未指定的。矢量不包括父字段名称。矢量中名称的顺序与在创建记录类型的定义记录类型窗体或 make-record-type-描述符调用中指定字段的顺序相同。

```
（定义记录类型点（字段 x y））
（定义记录类型 cpoint （父点） （字段颜色））
（记录类型字段名称
（记录类型描述符点）） ⇒ #（x y）
（记录类型字段名称
（记录类型描述符 cpoint）） ⇒ #（颜色）
```

过程：（记录字段可变？ rtd idx）
返回：#t如果指定的 rtd 字段是可变的，#f否则
库：（rnrs 记录检查）、（rnrs）

idx 必须是小于 rtd 字段数的非负整数，不计算父字段。idx 值 0 指定在创建记录类型的定义记录类型窗体或 make-record-type-描述符调用中给出的第一个字段，1 指定第二个字段，依此类推。

```
（定义记录类型点（字段 （可变 x） （可变 y））） 
（定义记录类型 cpoint（父点）（字段颜色））

（记录字段可变？（记录类型描述符点） 0） ⇒ #t
（记录字段可变？（记录类型描述符 cpoint） 0） ⇒ #f
```

程序：（记录？obj）

返回：如果 obj 是非不透明的记录实例，则#t，否则#f

库：（rnrs 记录检查）、（rnrs）

当传递不透明记录类型的实例时，record？ 返回#f。虽然不透明记录类型的实例实质上是记录，但不透明点是对程序中不应有权访问该信息的部分隐藏所有表示信息，这包括对象是否为记录。此外，此谓词的主要目的是允许程序检查是否有可能通过下面描述的记录-rtd 过程从参数中获取 rtd。

```
（定义记录类型语句（字段 str））
（定义 q （make-statement “He’s dead， Jim”））
（statement？ q） ⇒ #t
（record？ q） ⇒ #t

（define-record-type opaque-statement （fields str）
（opaque #t））
（define q （make-opaque-statement “He’s move on，
Jim”））
（opaque-statement？ q） ⇒ #t
（record？ q） ⇒ #f
```

过程：（记录 rtd 记录）

返回：记录

库的记录类型描述符 （rtd）：（rnrs 记录检查）、（rnrs）

该参数必须是非不透明记录类型的实例。结合本节和第 9.2 节中描述的一些其他过程，record-rtd 允许检查或更改记录实例，即使检查员不知道实例的类型也是如此。下面的过程打印字段说明了此功能，该过程打印字段接受记录参数并写入记录的每个字段的名称和值。

```
（定义打印字段
（lambda （r）
（除非 （record? r）
（断言违反 'print-fields “not a record” r））
（let loop （[rtd （record-rtd r）]）
（let （[prtd （record-type-parent rtd）]）
（when prtd （loop prtd））））
（let* （[v （record-type-field-names rtd）]
[n （vector-length v）]）
（do （[i 0 （+ i 1）]）
（ （= i n））
（write （vector-ref v i））
（display “=”）
（write （ （record-accessor rtd i） r））
（换行符））））））
```

使用熟悉的点和cpoint定义：

```
（定义记录类型点（字段 x y））
（定义记录类型的 cpoint（父点）（字段颜色））
```

表达式（打印字段 （make-cpoint -3 7 'blue））显示以下三行。

```
x=-3
y=7
颜色=蓝色
```

ISBN 978-0-262-51298-5 / LOC QA76.73.S34D93
订购本书 / 关于这本书

http://www.scheme.com