5
Метаклассы

Инициализация экземпляров

Пример метакласса

Иерархия наследования метаклассов

Инициализация переменных класса

Резюме о поиске метода

Словарь терминов

Поскольку все компоненты системы Smalltalk-80 представляются объектами, а все объекты являются экземплярами классов, сами классы тоже должны представляться экземплярами некоторых классов. Класс, экземпляры которого сами являются классами, называется *метаклассом*. Эта глава описывает характерные свойства метаклассов. На примерах иллюстрируется применение метаклассов для создания экземпляров и для общих запросов к классам.

В ранних версиях системы Smalltalk был только один метакласс, называвшийся Class (Класс). Такая организация изображена на рис. 5.1. Здесь, как и в главе 4, рамка обозначает класс, а кружок — экземпляр того класса, в котором кружок находится. Где необходимо, рамка отмечается именем класса, который она представляет. Заметим, что в рамке, помеченной именем Class, содержится столько кружков-экземпляров, сколько рамок-классов на рисунке.

Рис. 5.1.

Такой подход создавал трудности, связанные с тем, что протокол сообщений для всех классов был одним и тем же, поскольку был определен в одном месте. В частности, сообщения, используемые для создания новых экземпляров, были одинаковыми для всех классов и не могли учитывать каких-либо особых требований инициализации. В системе с единственным метаклассом все классы в ответ на сообщение new или new: возвращали экземпляр, все переменные которого ссылались на nil. Но для большинства объектов nil неприемлем в качестве значения переменной экземпляра, поэтому новые экземпляры приходилось инициализировать посылкой другого сообщения. Программист должен был следить, чтобы каждый раз после посылки сообщений new или new: новому объекту посылалось еще одно сообщение для выполнения инициализации. Примеры инициализации такого рода были приведены в главе 4 для классов SmallDictionary и FinancialHistory.

В системе Smalltalk-80 устранено ограничение, в связи с которым все классы имеют один и тот же протокол сообщений. По новой схеме каждый класс является экземпляром своего собственного метакласса. Теперь, всякий раз, когда создается новый класс, для него автоматически создается новый метакласс. Метаклассы похожи на другие классы, поскольку они содержат методы, используемые своими экземплярами. Метаклассы отличаются от других классов тем, что они не являются экземплярами своих метаклассов. Вместо этого, они все являются экземплярами класса с именем Metaclass (Метакласс). Кроме того, у метаклассов нет имен. Метакласс становится доступным, когда его экземпляру посылается унарное сообщения class. Например, на метакласс класса Rectangle можно сослаться выражением Rectangle class.

Сообщения метаклассам обычно посылаются для создания и инициализации экземпляров, а также для инициализации переменных класса.

Инициализация экземпляров

Каждый класс может отвечать на сообщения, которые требуют создания должным образом инициализированных новых экземпляров. Необходимо множество метаклассов, поскольку инициализационные сообщения различны для различных классов. Например, мы уже видели, что класс Time создает новый экземпляр в ответ на сообщение now, а класс Data создает новый экземпляр в ответ на сообщение today.

```
Time now
Date today
```

Но эти сообщения бессмысленны для класса Point, экземпляры которого представляют местоположение в двумерном пространстве. Класс Point создает новый экземпляр в ответ на сообщение с именем x:y: и двумя аргументами, задающими соответствующие координаты. Это сообщение, в свою очередь, бессмысленно для классов Time или Data.

```
Point x: 100 y: 150
```

Класс Rectangle понимает несколько сообщений создания новых экземпляров. Сообщение с именем origin:corner: использует аргументы-точки для обозначения верхнего левого и нижнего правого углов прямоугольника.

```
Rectangle origin: (Point x: 50 y: 50) corner: (Point x: 250 y: 300)
```

Сообщение с именем origin:extent: использует в качестве аргументов верхний левый угол и точку, которая представляет его ширину и высоту. Тот же самый прямоугольник может быть определен следующим выражением:

```
Rectangle origin: (Point x: 50 y: 50) extent: (Point x: 200 y: 250)
```

В системе Smalltalk-80 класс с именем Class — абстрактный суперкласс для всех метаклассов. Class характеризует общую природу классов. Каждый метакласс добавляет к ней специфическое поведение своего единственного экземпляра. Метаклассы могут добавлять новые сообщения для создания экземпляров, подобно упомянутым классам Data, Time, Point, Rectangle; они могут переопределять основные сообщения new и new: с тем, чтобы по умолчанию выполнять некоторую инициализацию.

Организация классов и метаклассов в системе проиллюстрирована на рис. 5.2.

Рис. 5.2.

На этом рисунке показаны классы Object, Metaclass, Class и метаклассы для каждого из них. Каждый кружок внутри рамки с пометкой Metaclass обозначает метакласс. Каждая рамка внутри рамки с пометкой Class обозначает подкласс класса Class. Существует одна такая рамка для каждого кружка в рамке с именем Metaclass. Каждая из этих рамок содержит кружок, обозначающий ее экземпляр; эти экземпляры ссылаются на класс Object или на один из его подклассов, но не на метаклассы.

Пример метакласса

Так как существует взаимно однозначное соответствие между классом и его метаклассом, их описания представляются совместно. Описание реализации класса включает в себя часть, называемую “class methods” (методы класса), в которой описываются методы, добавляемые метаклассом. Протокол для метакласса всегда можно найти, отыскав раздел class methods в описании реализации его единственного экземпляра. Таким образом, методы класса и методы экземпляра (instance methods) описываются вместе как части полного описания реализации класса. Следующая новая версия описания реализации класса FinancialHistory включает методы класса.

```
class name                  FinancialHistory
superclass                  Object
instance variable names     cashOnHand
                            incomes
                            expenditures

class methods
instance creation
    initialBalance: amount
        ↑super new setInitialBalance: amount

    new
        ↑super new setInitialBalance: 0
```

```
instance methods
transaction recording
    receive: amount from: source
        incomes   at: source
                  put: (self totalReceivedFrom: source) + amount.
        cashOnHand ← cashOnHand + amount

    spend: amount for: reason
        expenditures  at: reason
                      put: (self totalReceivedFrom: source) + amount.
        cashOnHand ← cashOnHand – amount

inquiries
    cashOnHand
        ↑cashOnHand

    totalReceivedFrom: source
        (incomes includesKey: source)
          ifTrue: [↑incomes at: source]
          ifFalse: [↑0]

    totalSpentFor: reason
        (expenditures includesKey: reason)
          ifTrue:  [↑expenditures at: reason]
          ifFalse: [↑0]

private
    setInitialBalance: amount
        cashOnHand ← amount.
        incomes ← Dictionary new.
        expenditures ← Dictionary new
```

Сделаны три изменения в описании реализации.

1. Добавлена одна категория методов класса, названная instance creation (создание экземпляра). Категория содержит методы initialBalance и new. По соглашению, категория instance creation используется для методов класса, которые возвращают новые экземпляры.
2. Исключена категория методов экземпляра, называемая initialization. Она включала в себя метод initialBalance:.
3. Добавлена категория методов экземпляра, названная private (частные). Категория содержит один метод setInitialBalance:, состоящий из тех же выражений, какие были в исключенном методе initialBalance:.

Этот пример показывает, как метаклассы создают инициализированные экземпляры. Методы создания экземпляров сообщениями initialBalance: и new не имеют прямого доступа к переменным нового экземпляра (cashOnHand, incomes и expenses) из-за того, что эти методы являются не частью класса, которому принадлежат новые экземпляры, а частью класса класса. Поэтому методы создания экземпляров сначала создают экземпляры с неинициализированными переменными, а потом посылают им сообщение инициализации setInitialBalance:. Метод для этого сообщения находится в описании реализации класса FinancialHistory уже среди методов экземпляра и этот метод "умеет" присваивать переменным экземпляра соответствующие значения. Сообщение инициализации не считается частью внешнего протокола класса FinancialHistory, а классифицируется как частное. Обычно оно посылается только один раз и только методом класса.

Старое сообщение инициализации initialBalance: было удалено, так как естественный способ создания экземпляра класса FinancialHistory — выражение

```
FinancialHistory initialBalance: 350
```

а не выражение

```
FinancialHistory new initialBalance: 350
```

В действительности, второе выражение теперь будет вызывать ошибку, так как экземпляр FinancialHistory больше не отвечает на сообщение initialBalance:. Можно было бы сохранить метод экземпляра для сообщения initialBalance: и одновременно реализовать для него метод класса, но мы стараемся не использовать одинаковые имена для методов экземпляра и класса с тем, чтобы сделать описание реализации более удобочитаемым. Однако, если бы и были использованы одинаковые имена, двусмысленности не возникло бы.

Иерархия наследования метаклассов

Как и другие классы, метакласс наследует информацию от суперкласса. Простейшим способом построить наследование метаклассов было бы сделать каждый метакласс подклассом класса Class. Такая организация была показана на рис. 5.2. Class описывает общую природу классов. Каждый метакласс добавляет особое поведение своему единственному экземпляру. Метаклассы могут добавлять новые сообщения для образования экземпляров и переопределять основные сообщения new и new: для необходимой инициализации по умолчанию.

Когда метаклассы были добавлены в систему Smalltalk-80, был сделан следующий шаг в организации классов. Теперь иерархия метаклассов образовывается параллельно иерархии классов, их экземпляров. Поэтому, если DeductibleHistory — подкласс класса FinancialHistory, то метакласс класса DeductibleHistory должен быть подклассом метакласса класса FinancialHistory. Метакласс, как правило, имеет только один экземпляр.

В системе существует абстрактный класс с именем ClassDescription (ОписаниеКласса) для описания классов и их экземпляров. Классы Class и Metaclass — подклассы класса ClassDescription. Поскольку цепочка суперклассов всех объектов заканчивается на классе Object, а сам класс Object не имеет суперкласса, метакласс класса Object имеет своим суперклассом класс Class. Тем самым, от класса Class все метаклассы наследуют сообщения, которые обеспечивают протокол для создания экземпляров (см. рис. 5.3).

Цепочка суперклассов от класса Class приводит в конечном счете к классу Object. Заметим, что иерархия рамок внутри рамки с пометкой Object class такая же, как и иерархия рамок внутри рамки с пометкой Object; это сходство показывает параллельность двух иерархий. Полное описание этой части системы, включая связь между классом Metaclass и его метаклассом, будет рассматриваться в главе 16.

Рис. 5.3.

Как пример наследования в иерархии метаклассов рассмотрим реализацию сообщения initialBalance: в классе FinancialHistory class.

```
initialBalance: amount
    ↑super new setInitialBalance: amount
```

Данный метод создает новый экземпляр, выполняя выражение super new. Метод для сообщения new ищется среди методов класса суперкласса, а не среди методов класса FinancialHistory. Затем новому экземпляру посылается сообщение setInitialBalance:, с начальной суммой баланса в качестве значения аргумента amount. Подобным образом переопределяется и сообщение new, создавая новый экземпляр выражением super new и затем посылая ему сообщение setInitialBalance:.

```
new
    ↑super new setInitialBalance: 0
```

Где же в действительности находится метод для сообщения new, которое посылается super? Иерархия метаклассов параллельна иерархии их экземпляров: если один класс —

подкласс второго класса, то его метакласс — подкласс метакласса второго класса (см. рис. 5.3). Параллельность иерархий классов и метаклассов применительно к приложению FinancialHistory показана на рис. 5.4.

Рис. 5.3.

При выполнении выражения

```
FinancialHistory initialBalance: 350
```

поиск ответа на сообщение initialBalance: начинается в FinancialHistory class, то есть среди методов класса FinancialHistory, поскольку сообщение послано классу. Метод для этого имени там существует и содержит два сообщения:

1. Послать переменной super сообщение new.
2. Послать полученному результату сообщение setInitialBalance: 350.

Поиск метода для сообщения new начинается в суперклассе для FinancialHistory class, а именно, в Object class. Нужного метода там нет и поиск продолжается по иерархии в следующем суперклассе, то есть в классе Class. Нужный метод для сообщения new находится в классе Class и выполняется (это примитивный метод системы). Результат — неинициализированный экземпляр класса FinancialHistory. Этому экземпляру посылается сообщение setInitialBalance:. Так как сообщение послано экземпляру класса FinancialHistory, то поиск метода для этого сообщения начинается в классе FinancialHistory среди методов экземпляра. Там нужный метод есть, он выполняется, присваивая значение каждой переменной экземпляра.

Выполнение выражения

```
FinancialHistory new
```

происходит аналогично. Метод для сообщения new находится в классе FinancialHistory class (то есть среди методов класса FinancialHistory). Последующие действия те же самые, что и для initialBalance:, за исключением значения аргумента в setInitialBalance:. Методы создания экземпляров должны обязательно использовать сообщение super new, чтобы избежать рекурсивного вызова самих себя.

Инициализация переменных класса

Другое важное применение сообщений классам, отличное от инициализации экземпляров, — инициализация переменных класса. Объявления переменных в описании реализации задают только имена переменных класса, но не их значения. Когда класс создается, создаются именованные переменные класса, но все они получают значение nil. Обычно метакласс определяет метод, который инициализирует переменные класса. По соглашению, метод инициализации переменных класса связывается с унарным сообщением initialize из категории class initialization (инициализация класса).

Переменные класса доступны как классу, так и его метаклассу. Присваивание значений переменным класса может быть сделано в методах класса, а не косвенно через частное сообщение в методах экземпляра (как это было необходимо для переменных экземпляра).

Ниже приводится пример класса DeductibleHistory с переменной класса, которая должна быть инициализирована. Класс DeductibleHistory — это подкласс класса FinancialHistory. Он определяет одну переменную класса — MinimumDeductions.

class name	DeductibleHistory
superclass	FinancialHistory
instance variable names	deductibleExpenditures
class variable names	MinimumDeductions

```
class methods
instance creation
    initialBalance: amount
        | newHistory |
        newHistory ← super initialBalance: amount.
        newHistory initializeDeductions.
        ↑newHistory

    new
        | newHistory |
        newHistory ← super initialBalance: 0.
        newHistory initializeDeductions.
        ↑newHistory

class initialization
    initialize
        MinimumDeductions ← 2300

instance methods
transaction recording
    spendDeductible: amount for: reason
        self spend: amount for: reason.
        deductibleExpenditures ← deductibleExpenditures + amount

    spend: amount for: reason deducting: deductibleAmount
        self spend: amount for: reason.
        deductibleExpenditures ← deductibleExpenditures + deductialeAmount

inquiries
    isItemizable
        ↑deductibleExpenditures >= MinimumDeductions

    totalDeductions
        ↑deductibleExpenditures

private
    initializeDeductions
        deductibleExpenditures ← 0
```

Данная версия класса DeductibleHistory добавляет пять методов экземпляра, один из которых — isItemizable. Ответ на это сообщение — true или false, в зависимости от того, достаточно ли было накоплено налогооблагаемых сумм, чтобы составлять их перечень в отчете о налогах. Закон о налогах определяет, что минимальная облагаемая налогом сумма составляет 2300, так что, если накоплено меньше, то должен быть использован стандартный налог. Именно на константу 2300 ссылается переменная класса MinimumDeductions. Для того чтобы с успехом можно было посылать экземпляру класса DeductibleHistory сообщение isItemizable, переменной класса MinimumDeductions должно быть присвоено ее числовое значение. Это достигается посылкой классу сообщения initialize еще до того, как будет создан его первый экземпляр.

```
DeductibleHistory initialize
```

Такое сообщение посылается только один раз, после того, как впервые определяется сообщение инициализации класса. Переменная класса доступна каждому новому экземпляру этого класса.

В соответствии с приведенным выше описанием класса, новый экземпляр класса DeductibleHistory может быть создан посредством посылки классу сообщения

initialBalance: или new, точно также, как и для его суперкласса FinancialHistory. Допустим, что мы выполняем выражение

```
DeductibleHistory initialBalance: 100
```

Определение того, какие методы действительно реализуются при выполнении выражения, зависит от цепочки "класс/суперкласс" для класса DeductibleHistory. Метод для сообщения initialBalance: находится среди методов класса DeductibleHistory.

```
initialBalance: amount
    | newHistory |
    newHistory ← super initialBalance: amount.
    newHistory initializeDeductions.
    ↑newHistory
```

Метод определяет переменную newHistory как временную переменную. Первое выражение метода — присваивание значения временной переменной.

```
newHistory ← super initialBalance: amount
```

Псевдопеременная super ссылается на получателя сообщения. Получатель — класс DeductibleHistory; его класс — это его метакласс. Суперкласс метакласса — метакласс класса FinancialHistory. Таким образом, мы сможем найти метод, который надо выполнить, просматривая методы класса в FinancialHistory. Это метод

```
initialBalance: amount
    ↑super new setInitialBalance: amount
```

Мы уже проследили ранее за его выполнением. Ответ на сообщение new находится в классе Class. Создается новый экземпляр первоначального получателя сообщения, класса DeductibleHistory, и ему посылается сообщение setInitialBalance:. Поиск метода setInitialBalance: начинается в классе нового экземпляра, то есть в классе DeductibleHistory. Но там нужного метода нет. Поиск продолжается в его суперклассе FinancialHistory. Здесь он находится и выполняется. Переменным экземпляра, определенным в FinancialHistory, присваиваются значения. Таким образом, значение первого выражения метода класса initialBalance: из класса DeductibleHistory — частично инициализированный новый экземпляр, который становиться значением временной переменной newHistory.

Затем переменной newHistory посылается сообщение initialDeductions. Поиск начинается в классе получателя сообщения, то есть в классе DeductibleHistory. Там нужный метод есть. Он присваивает четвертой переменной экземпляра значение 0.

Наконец, третье выражение метода, связанного с сообщением создания нового экземпляра, возвращает новый экземпляр.

Альтернативный способ реализации класса DeductibleHistory приведен ниже. В нем методы класса из класса FinancialHistory, создающие экземпляры, не переопределяются. Но, поскольку изменилось количество переменных экземпляра, переопределяется метод для частного сообщения setInitialBalance:.

```
class name                  DeductibleHistory
superclass                  FinancialHistory
instance variable names     deductibleExpenditures
class variable names        MinimumDeductions

class methods
class initialization
    initialize
        MinimumDeductions ← 2300

instance methods
```

```
transaction recording
    spendDeductible: amount for: reason
        self spend: amount for: reason.
        deductibleExpenditures ← deductibleExpenditures + amount

    spend: amount for: reason deducting: deductibleAmount
        self spend: amount for: reason.
        deductibleExpenditures <- deductibleExpenditures + deductialeAmount

inquiries
    isItemizable
        ↑deductibleExpenditures >= MinimumDeductions

    totalDeductions
        ↑deductibleExpenditures

private
    setInitialBalance: amount
        super setInitialBalance: amount.
        deductibleExpenditures ← 0
```

С учетом альтернативного описания класса DeductibleHistory, получение ответа на сообщение initialBalance: в выражении

```
DeductibleHistory initialBalance: 350
```

начнется с поиска метода initialBalance: в классе DeductibleHistory class. Его там нет. Продолжается поиск в классе FinancialHistory class. Там он есть. Выполняемый метод содержит выражение

```
super new setInitialBalance: amount
```

Метод new определен в классе Class. Поиск setInitialBalance: начинается в классе нового экземпляра, то есть в классе DeductibleHistory. Метод setInitialBalance: определен в классе DeductibleHistory. Выполнение сообщения setInitialBalance: из класса DeductibleHistory начинается с того, что точно такое же сообщение посылается псевдопеременной super, поэтому поиск соответствующего метода начинается в классе FinancialHistory. Такой метод там есть и трем переменным экземпляра присваиваются значения. Второе выражение в методе setInitialBalance: из класса DeductibleHistory устанавливает четвертую переменную в 0. В результате выполнения первоначального выражения создается полностью инициализированный экземпляр класса DeductibleHistory.

Резюме о поиске метода

Определение того, какие действия реально будут выполнены в ответ на посланное объекту сообщение, включает поиск метода в иерархии классов получателя. Поиск начинается с класса получателя и продолжается по цепочке “класс-суперкласс”. Если метод не найден в последнем суперклассе — классе Object, то сообщается об ошибке. Если получатель сообщения — класс, то его класс — метакласс. Сообщения, на которые может ответить класс, перечисляются в части описания реализации класса, называемой class methods. Если получатель сообщения — не класс, то сообщения, на которые он может ответить, перечисляются в части описания реализации класса, называемой instance methods.

Псевдопеременная self ссылается на получателя сообщения, которое вызвало выполняемый метод. Поиск метода, соответствующего сообщению к self, начинается в классе получателя. Псевдопеременная super также ссылается на получателя сообщения. Однако

поиск метода, соответствующего сообщению к super, начинается в суперклассе класса, в котором был найден выполняемый метод.

Этим заканчивается описание языка программирования Smalltalk-80. Чтобы пользоваться системой Smalltalk-80, программист должен иметь общее представление о ее системе классов. Во второй части книги дается детальное описание протокола для каждого класса системы и приводятся многочисленные примеры, часто представляющие описание реализации системных классов. В третьей части книги описываются приложения. Перед тщательным изучением существующей системы классов, читатель мог бы, при желании, сначала обратиться к части 3, чтобы получить представление о том, как применять язык Smalltalk-80 для разработки больших приложений.

Словарь терминов

метакласс	класс класса
Class	абстрактный суперкласс всех классов, таких как метаклассы.
Metaclass	класс, чьи экземпляры есть классы классов.