9
Протокол для всех классов наборов

Объекты и сообщения

Классы и экземпляры

Пример приложения

Системные классы

Арифметика

Структуры данных

Управляющие структуры

Среда программирования

Просмотр и взаимодействие

Коммуникации

Словарь терминов

Литералы

Числа

Символы

Строки

Имена

Массивы

Переменные

Присваивания

Имена псевдопеременных

Сообщения

Имена сообщений и аргументы

Возвращение значений

Синтаксический анализ

Соглашения о форматировании

Каскадирование

Блоки

Управляющие структуры

Условные выражения

Аргументы блока

Словарь терминов

Описание протокола

Категории сообщений

Описание реализации

Объявление переменных

Переменные экземпляра

Общие переменные

Методы

Имена аргументов

Возвращаемые значения

Псевдопеременная self

Временные переменные

Примитивные методы

Словарь терминов.

Описание подкласса

Пример подкласса

Выбор метода

Сообщения псевдопеременной self

Сообщения псевдопеременной super

Абстрактные суперклассы

Системные сообщения для работы с подклассами

Словарь терминов

Инициализация экземпляров

Пример метакласса

Иерархия наследования метаклассов

Инициализация переменных класса

Резюме о поиске метода

Словарь терминов

Проверка функциональности объекта

Сравнение объектов

Копирование объектов

Доступ к частям объекта

Печать и сохранение объектов

Обработка ошибок

Класс Magnitude

Класс Date

Класс Time

Класс Character

Протокол числовых классов

Классы Float **и** Fraction

Случайный выбор и карточные игры

Проблема пьяного таракана

Обход бинарных деревьев

Бинарное дерево слов

Класс Stream

Позиционируемые потоки

Класс ReadStream

Класс WriteStream

Класс ReadWriteStream

Потоки генерируемых элементов

Потоки для наборов без внешних ключей

Внешние потоки и файловые потоки

Класс Collection

Подклассы класса Collection

Класс Bag

Класс Set

Класс Dictionary

Класс SequenceableCollection

Подклассы класса SequenceableCollection

Класс MappedCollection

Класс UndefinedObject

Классы Boolean, True **и** False

Дополнительный протокол класса Object

Отношения зависимости между объектами

Обработка сообщений

Сообщения системных примитивов

Процессы

Планирование

Приоритеты

Семафоры

Взаимное исключение

Разделение ресурсов

Аппаратные прерывания

Класс SharedQueue

Класс Delay

Класс Behavior

Класс ClassDescription

Класс Metaclass

Класс Class

Окна

Выбор текста

Выбор из меню

Окна просмотра

Выбор из списка

Прокрутка

Определение класса

Тестирование

Инспекторы

Сообщения об ошибках

Окно извещений

Класс DisplayObject

Класс DisplayMedium

Формы

Другие формы

Курсоры

Класс DisplayScreen

Класс DisplayText

Пути

Преобразование форм

Увеличение

Вращение

Заполнение области

Игра “Жизнь”

Компилятор

Скомпилированные методы

Байткоды

Интерпретатор

Контексты

Контексты блоков

Сообщения

Примитивные методы

Память объектов

Аппаратные средства и дополнительные системные классы

Основные понятия распределения вероятностей

Определения

Простые примеры

Класс ProbabilityDistribution

Класс DiscreteProbability

Класс ContinuousProbability

Дискретные распределения вероятностей

Распределение Бернулли

Биномиальное распределение

Геометрическое распределение

Распределение Пуассона

Непрерывные распределения вероятностей

Равномерное распределение

Экспоненциальное распределение

Гамма-распределение

Нормальное распределение

Основные понятия компьютерного моделирования

Объекты модели

Модели

Пример "по умолчанию": класс NothingAtAll

Реализация классов моделирования

Класс SimulationObject

Класс DelayedEvent

Класс Simulation

Трассировка примера NothingAtAll

Временная статистика

Гистограммы пропускной способности

Подсчет событий

Наблюдение за событиями

Реализация классов ResourcesProvider **и** WaitingSimulationObject

Расходуемые ресурсы

Нерасходуемые ресурсы

Пример: файловая система

Возобновляемые ресурсы

Пример: обслуживание паромной переправы

Реализация класса ResourceCoordinator

Пример: модель мойки автомобилей

Пример: модель паромной переправы для спецгрузовиков

Пример: банк

Пример: информационная система

Набор — это группа объектов, называемых *элементами* набора. Примером набора может быть массив. Например, массив

```
#('word' 3 5 $G (1 2 3))
```

состоит из пяти элементов. Первый — строка, второй и третий — числа, четвертый — символ, пятый — массив. Первый элемент — строка — тоже набор, в данном случае, состоящий из четырех символов.

Наборы предоставляют основные структуры данных для программирования в системе Smalltalk-80. Элементы одних наборов могут быть упорядочены, других — не упорядочены. Некоторые неупорядоченные наборы, например, экземпляры класса Bag, допускают дублирование элементов в наборе, а другие, например, экземпляры класса Set, этого не допускают. Есть также словари (экземпляры класса Dictionary), которые в качестве элементов содержат ассоциативные пары объектов. Среди упорядоченных наборов некоторые определяют порядок добавления элементов в набор внешними причинами (экземпляры класса OrderedCollection, Array, String), другие определяют порядок, опираясь на свойства самих элементов (экземпляры класса SortedCollection). Например, такие общеупотребительные структуры данных, как массивы и строки, представляются классами, которые связывают свои элементы с целочисленными индексами и определяют внешний порядок элементов в соответствии с порядком индексов.

В этой главе вводится общий протокол для всех наборов. Каждое описанное здесь сообщение понятно любому виду набора, за исключением тех, которые намеренно отменяют это сообщение. Описание различных видов наборов будет приведено в следующей главе.

Наборы поддерживают четыре категории сообщений для доступа к элементам:

- сообщения для добавления новых элементов
- сообщения для удаления элементов
- сообщения для проверки принадлежности элемента набору
- сообщения для перечисления элементов

Элемент или группу элементов можно добавить или удалить из набора. Можно узнать, пуст ли набор, включает ли он указанный элемент? Можно определить, сколько раз данный элемент входит в набор. Перечисление позволяет получить доступ к элементам без удаления их из набора.

Добавление, удаление и проверка элементов

Основной протокол для наборов определяется суперклассом всех классов наборов, который называется Collection (Набор). Класс Collection — подкласс класса Object. Протокол добавления, удаления и проверки элементов следующий.

Collection instance protocol

adding	
add: newObject	Включает аргумент newObject как один из элементов получателя. Возвращает newObject.
addAll: aCollection	Включает все элементы аргумента aCollection как элементы получателя. Возвращает aCollection.
removing	
remove: oldObject	Удаляет аргумент oldObject как элемент получателя. Возвращает oldObject. Если в получателе нет элемента, равного oldObject, выдает сообщение об ошибке.

remove: oldObject ifAbsent: anExceptionBlock
: Удаляет аргумент oldObject как элемент получателя. Если несколько элементов равны oldObject, удаляет первый найденный. Возвращает oldObject. Если в получателе нет элемента, равного oldObject, возвращает результат выполнения блока anExceptionBlock.

removeAll: aCollection
: Удаляет все элементы аргумента aCollection из получателя. Возвращает aCollection, если удаление всех элементов прошло успешно, иначе выдает сообщение об ошибке.

testing

includes: anObject
: Возвращает true, если один из элементов получателя равен аргументу anObject, иначе возвращает false.

isEmpty
: Возвращает true, если в получателе нет элементов, иначе возвращает false.

occurencesOf: anObject
: Возвращает число элементов получателя, равных аргументу anObject.

Для того чтобы продемонстрировать эти сообщения, введем в рассмотрение набор lotteryA, равный

(272 572 852 156)

и набор lotteryB, равный

(572 621 274)

Предположим, что эти два набора, представляющие числа, выпавшие при розыгрыше лотереи — экземпляры класса Bag (ПростойНабор), подкласса Collection. Сам класс Collection абстрактный, в том смысле, что в нем описывается общий протокол для всех наборов. Он не достаточно конкретизирован, чтобы обеспечить хранение элементов набора, и потому не может реализовать все свои сообщения. Эта неполнота описания не позволяет классу Collection создавать экземпляры. Класс Bag — конкретный, в том смысле, что он обеспечивает представление для хранения элементов и реализует все сообщения, не реализуемые в суперклассе.

Все наборы отвечают на сообщение size, возвращая количество элементов в наборе. Так, мы можем определить, что

lotteryA size

есть 4, а

lotteryB size

есть 3. Выполняя по порядку следующие сообщения, получим:

выражение	*результат*	lotteryA, *если он изменялся*
lotteryA isEmpty	false	
lotteryA includes: 572	true	
lotteryA add: 596	596	Bag(272 572 852 156 596)
lotteryA addAll: lotteryB	Bag(572 621 274)	Bag(272 274 852 156 596 572 572 621)
lotteryA occurencesOf: 572	2	
lotteryA remove: 572	572	Bag(272 274 852 156 596 572 621)
lotteryA size	7	
lotteryA removeAll: lotteryB	Bag(572 621 274)	Bag(272 852 596 156)
lotteryA size	4	

Заметим, что сообщения add: и remove: возвращают аргументы, а не сам набор, так что можно получить доступ к вычисленному аргументу. Сообщение remove: удаляет только один, а не все входящие в набор элементы, равные аргументу.

Блоки были введены в главе 2. Сообщение remove: oldObject ifAbsent: anExceptionBlock (удалить: старыйОбъект еслиОтсутствует: особыйБлок) использует блок для того, чтобы определить поведение набора в случае возникновения ошибки. Если в получателе нет элемента, ссылающегося на oldObject, выполняется блок-аргумент anExceptionBlock. Такой блок может, например, содержать обработку ошибки, или может просто проигнорировать ее. Так, выражение

```
lotteryA remove: 121 ifAbsent: []
```

ничего не делает, после того, как определяет, что числа 121 нет в наборе lotteryA .

По умолчанию, сообщение remove: посылает набору-получателю сообщение error: 'object is not in the collection' (ошибка: 'объекта в наборе нет'). Напомним, что сообщение error: определено в классе Object, а потому понимается и всеми наборами.

Перечисление элементов

Протокол экземпляра для наборов включает несколько сообщений перечисления, которые дают возможность перебрать все элементы набора и использовать каждый из них при выполнении блока. Основное сообщение перечисления — do: aBlock (выполнить: aBlock). В нем аргументом служит одноаргументный блок aBlock, который выполняется один раз для каждого элемента из набора. Пусть letters — некоторый набор символов и мы хотим знать, сколько раз в нем встречаются символы a и A.

```
count ← 0.
letters do: [:each | each asLowercase == $a
                    ifTrue: [count ← count + 1]]
```

Здесь происходит увеличение счетчика count на 1 всякий раз, когда элемент из набора letters оказывается совпадающим с символом “a” в верхнем или нижнем регистре. Желаемый результат равен окончательному значению счетчика. Мы можем использовать идентичность == вместо равенства =, так как все объекты класса Character уникальны.

В классе Collection определены шесть основных сообщений перечисления. В описании этих сообщений говорится, что для сохранения результатов создается “новый набор, подобный получателю сообщения”. Эта фраза означает, что новый набор — экземпляр того же самого класса, что и получатель. Например, если получатель сообщения select: — экземпляр класса Set или Array, то ответ — новый экземпляр класса Set или Array, соответственно. В системе Smalltalk-80 есть только одно исключение — в реализации класса Interval, когда в ответ на сообщение перечисления возвращается массив, а не новый экземпляр класса Interval. Причина этого исключения в том, что элементы экземпляра класса Interval создаются в момент первоначального создания самого интервала; невозможно поместить какие бы то ни было элементы внутри уже существующего интервала.

Collection instance protocol

enumerating	
do: aBlock	Выполняет аргумент aBlock для каждого элемента получателя.
select: aBlock	Выполняет аргумент aBlock для каждого элемента получателя. Собирает в новый набор, подобный получателю, только те элементы, для которых блок выдает значение true. Возвращает новый набор.

reject: aBlock
: Выполняет аргумент aBlock для каждого элемента получателя. Собирает в новый набор, подобный получателю, только те элементы, для которых блок выдает значение false. Возвращает новый набор.

collect: aBlock
: Выполняет аргумент aBlock для каждого элемента получателя. Возвращает новый набор, подобный получателю, содержащий значения блока при каждом его выполнении.

detect: aBlock
: Выполняет аргумент aBlock для каждого элемента получателя. Возвращает первый элемент, для которого блок выдает значение true. Если такого значения нет, сообщает об ошибке.

detect: aBlock ifNone: exceptionBlock
: Выполняет аргумент aBlock для каждого элемента получателя. Возвращает первый элемент, для которого блок выдает значение true. Если такого значения нет, выполняет аргумент exceptionBlock, который должен быть блоком без аргументов.

inject: thisValue into: binaryBlock
: Выполняет аргумент binaryBlock один раз для каждого элемента получателя. Блок binaryBlock должен иметь два аргумента: второй аргумент — элемент из получателя; первый аргумент — значение предыдущего выполнения им блока, начиная с аргумента thisValue. Возвращает последний результат выполнения блока.

Каждое сообщение перечисления дает эффективный способ для представления последовательности сообщений, проверяющих или собирающих информацию об элементах набора.

Выбор и исключение

Мы могли бы определить количество вхождений символов a или A в набор letters, используя сообщение select:

```
(letters select: [:each | each asLowercase == $a]) size
```

То есть, создать новый набор, содержащий только те элементы letters, которые равны a или A, а затем вычислить размер этого набора.

Мы могли бы получить тот же самый результат, используя сообщение reject:

```
(letters reject: [:each | each asLowercase ~~ $a]) size
```

То есть, создать новый набор, исключая из letters элементы не равные a или A, а затем вычислить размер этого набора.

При выборе между сообщениями select: и reject: следует опираться на наилучшее выражение отбора элементов. Если выбор элементов лучше выражается в терминах принадлежности набору, то следует предпочесть сообщение select:, если же выбор лучше выражается в терминах отбраковки неподходящих элементов, то следует предпочесть сообщение reject:. В приведенном примере сообщение select: на наш взгляд предпочтительнее.

Еще один пример. Предположим, что employees — набор рабочих предприятия, каждый из которых отвечает на сообщение salary о размере его заработной платы. Создать набор работников, чья зарплата составляет по крайней мере $10.000, можно так:

```
employees select: [:each | each salary >= 10000]
```

или так:

```
employees reject: [ :each | each salary < 10000]
```

Результирующие наборы будут одинаковы. Выбор между сообщениями select: и reject: зависит от того, как программист захочет выразить критерий “по крайней мере $10.000”.

Сбор

Предположим, нам необходимо составить набор, в котором каждый элемент — заработок каждого рабочего из набора employees.

```
employees collect: [ :each | each salary]
```

Результирующий набор будет иметь тот же размер, что и набор employees. Каждый элемент нового набора — заработок соответствующего рабочего из набора employees.

Обнаружение

Предположим, мы хотим найти хотя бы одного рабочего с заработком, превышающим $20.000. Выражение

```
employees detect: [:each | each salary > 20000]
```

вернет нужный элемент из набора employees, если таковой имеется. Если такого элемента нет, набору employees будет послано сообщение error: 'object is not in the collection' ('объекта нет в наборе'). Точно так же, как и при определении сообщений удаления, программист имеет возможность задать исключительное поведение в случае отсутствия нужного элемента. Следующее выражение либо вернет ссылку на рабочего, чей заработок превышает $20.000, либо, если такого нет, вернет nil.

```
employees detect: [:each | each salary > 20000] ifNone: [nil].
```

Вставка

В сообщении inject:into: (вставить:в:) первый аргумент — начальное значение, которое участвует в определении конечного результата; второй аргумент — блок с двумя аргументами. Первый аргумент блока — переменная, которая ссылается на результат предыдущего выполнения блока, второй аргумент блока по очереди ссылается на каждый элемент набора. Используем это сообщение для вычисления суммарного заработка всех рабочих из набора employees.

```
employees inject: 0
        into: [:subTotal :nextWorker | subTotal + nextWorker salary]
```

В этом выражении первоначальное значение суммы 0 увеличивается на каждом шаге вычислений на величину заработка очередного рабочего из employees. Окончательный результат — последнее значение переменной subTotal.

Используя сообщение inject:into:, программист локально определяет имена временных переменных и избегает явной инициализации объектов, в которых накапливаются результаты. Например, в ранее приведенном примере подсчета числа символов из набора letters, совпадающих с a или A, мы вводили счетчик count.

```
count ← 0.
letters do: [:each | each asLowercase == $a
                ifTrue: [count ← count + 1]]
```

Другой подход использует сообщение inject:into:. Как и раньше, мы накапливаем результат в переменной count. Начальное значение count равно нулю. Если следующий символ в наборе letters равен a или A, то к count добавляется 1, иначе добавляется 0.

```
letters    inject: 0
           into: [:count :nextElement |
                count + (nextElement asLowercase == $a
                     ifTrue: [1]
                     ifFalse: [0])].
```

Создание экземпляров

В начале этой главы приводились примеры наборов, которые выражались как литералы. Это были экземпляры классов Array и String. Например, выражение для создания массива имеет вид

```
#('first' 'second' 'third')
```

где каждый элемент массива — это литерально задаваемая строка.

Для создания экземпляров конкретного вида наборов можно применять сообщения new и new:. Дополнительно, протокол класса для всех наборов содержит сообщения, обеспечивающие создание нового набора с одним, двумя, тремя или четырьмя элементами. Эти сообщения обеспечивают краткую запись для создания тех видов наборов, которые не допускают литеральных представлений.

Collection class protocol

instance creation

with: anObject	Возвращает набор с одним элементом anObject.
with: firstObject with: secondObject	Возвращает набор с двумя элементами firstObject и secondObject.
with: firstObject with: secondObject with: thirdObject	Возвращает набор с тремя элементами firstObject, secondObject и thirdObject.
with: firstObject with: secondObject with: thirdObject with: fourthObject	Возвращает набор с элементами firstObject, secondObject, thirdObject и fourthObject.

Например, класс Set — подкласс класса Collection. Создать множество с тремя элементами, символами s, e и t, можно, если выполнить выражение

```
Set with: $s with: $e with: $t
```

Заметим, что основная причина, по которой только эти четыре сообщения создания экземпляров (а не больше и не меньше) обеспечиваются протоколом класса, состоит в том, что именно такое их количество оказалось достаточным для того, чтобы создать наборы используемые самой системой.

Преобразование между классами наборов

Полное описание и понимание допустимых преобразований между различными видами наборов зависит от способа представления подклассов класса Collection. Здесь мы только отметим, что в протоколе преобразования для всех наборов определено пять сообщений, позволяющих привести получателя к виду экземпляра класса Bag, Set, OrderedCollection или SortedCollection. Эти сообщения определены в классе Collection для того, чтобы можно было преобразовать любой набор в экземпляр любого из указанных классов. При преобразовании набора неупорядоченных элементов в набор упорядоченных элементов, результирующий порядок элементов последнего будет произвольным.

Collection instance protocol

converting

asBag	Возвращает экземпляр класса Bag с теми же самыми элементами, что и у получателя.

asSet	Возвращает экземпляр класса Set с теми же самыми элементами, что и получателя (при этом дубликаты уничтожаются).
asOrderedCollection	Возвращает экземпляр класса OrderedCollection с теми же самыми элементами, что и у получателя (упорядочивание элементов может быть произвольным).
asSortedCollection	Возвращает экземпляр класса SortedCollection с теми же самыми элементами, что и у получателя, отсортированными так, что каждый элемент меньше или равен последующего (<=).
asSortedCollection: aBlock	Возвращает экземпляр класса SortedCollection с теми же самыми элементами, что и у получателя, отсортированными согласно аргументу aBlock.

Например, если lotteryA — экземпляр класса Bag с элементами

```
272 572 852 156 596 272 572
```

то набор

```
lotteryA asSet
```

будет множеством состоящим из элементов

```
852 596 156 572 272
```

а набор

```
lotteryA asSortedCollection
```

будет экземпляром класса SortedCollection, содержащим упорядоченные элементы (первый элемент — крайний слева):

```
156 272 272 572 572 596 852
```