1. Stream API的理解：
   1. Stream關注的是對數據的運算，與CPU打交道  
      集合關注的是數據的存儲，與內存打交道
   2. Java8提供了一套api，使用這套api可以對內存中的數據進行過濾、排序、映射、規約等操作。類似於sql對數據庫中表的相關操作。
2. 注意點：

① Stream 自己不會存儲元素。

② Stream 不會改變源對象。相反，他們會返回一個持有結果的新Stream。  
③ Stream 操作是延遲執行的。這意味著他們會等到需要結果的時候才執行

1. Stream的使用流程：

① Stream的實例化  
② 一系列多個的中間操作（過濾、映射、...)  
③ 終止操作

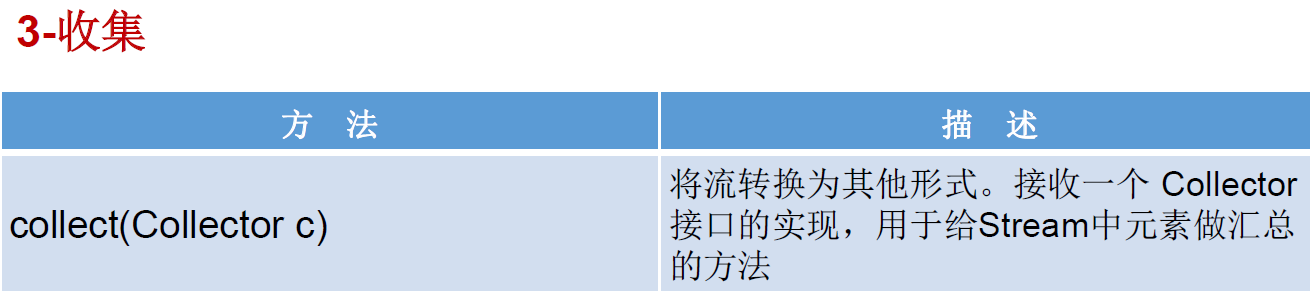
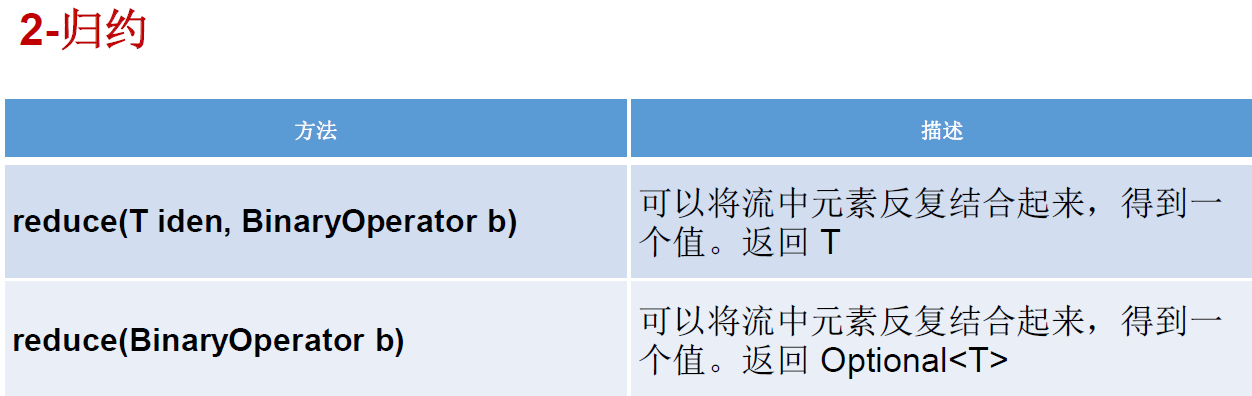
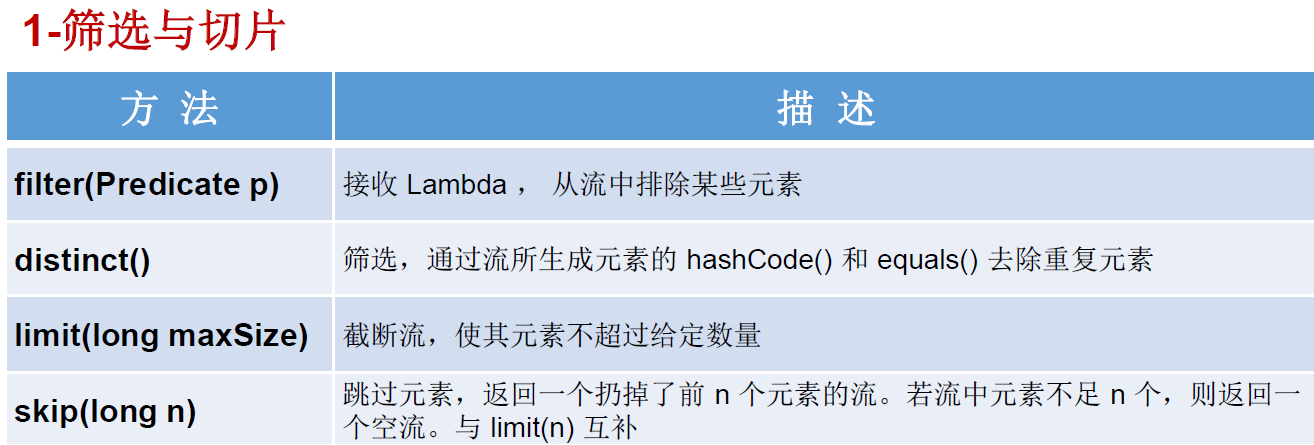
1. 使用流程的注意點：

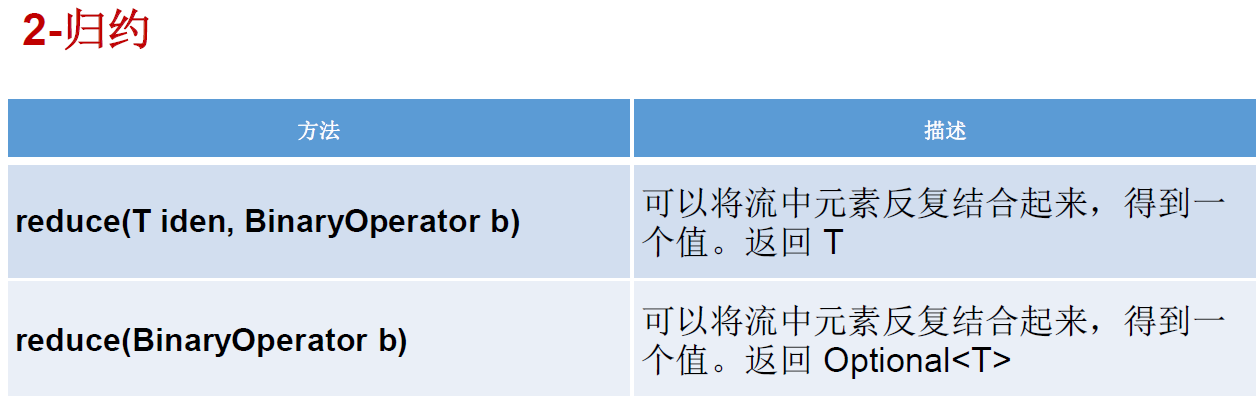
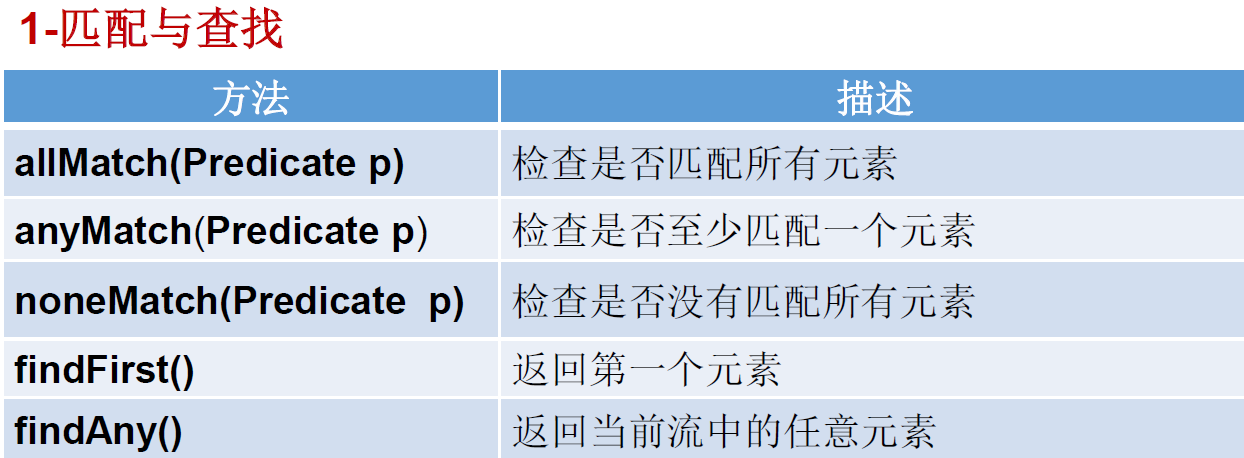
4.1 一個中間操作鏈，對數據源的數據進行處理  
4.2 一旦執行終止操作，就執行中間操作鏈，並產生結果。之後，不會再被使用

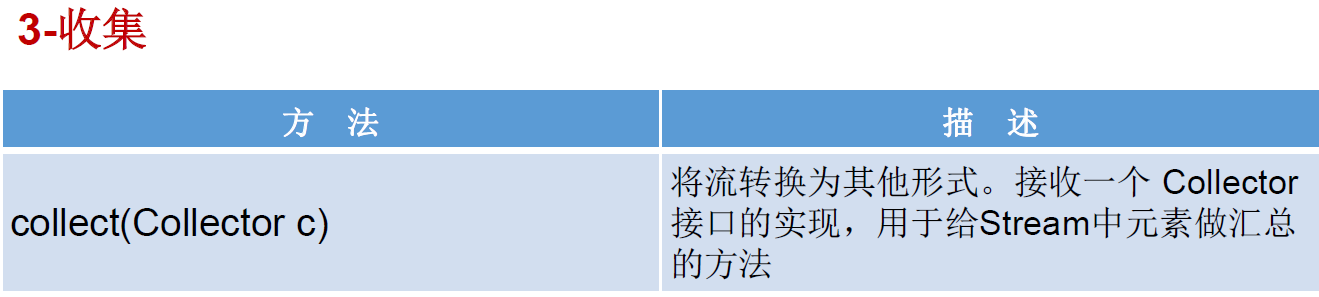
1. 步驟一：Stream實例化

*//創建 Stream方式一：通過集合* @Test  
 public void test1*(){* List*<*Employee*>* employees = EmployeeData.*getEmployees()*;  
*// default Stream<E> stream() : 返回一個順序流* Stream*<*Employee*>* stream = employees.stream*()*;  
*// default Stream<E> parallelStream() : 返回一個並行流* Stream*<*Employee*>* parallelStream = employees.parallelStream*()*;  
 *}  
 //創建 Stream方式二：通過數組* @Test  
 public void test2*(){* int*[]* arr = new int*[]{*1,2,3,4,5,6*}*;  
 *//調用Arrays類的static <T> Stream<T> stream(T[] array): 返回一個流* IntStream stream = Arrays.*stream(*arr*)*;  
 Employee e1 = new Employee*(*1001,"Tom"*)*;  
 Employee e2 = new Employee*(*1002,"Jerry"*)*;  
 Employee*[]* arr1 = new Employee*[]{*e1,e2*}*;  
 Stream*<*Employee*>* stream1 = Arrays.*stream(*arr1*)*;  
 *}*

*//創建 Stream方式三：通過Stream的of()* @Test  
 public void test3*(){* Stream*<*Integer*>* stream = Stream.*of(*1, 2, 3, 4, 5, 6*)*;  
 *}  
 //創建 Stream方式四：創建無限流* @Test  
 public void test4*(){  
// 叠代  
// public static<T> Stream<T> iterate(final T seed, final UnaryOperator<T> f)  
 //遍歷前10個偶數* Stream.*iterate(*0, t -> t + 2*)*.limit*(*10*)*.forEach*(*System.*out*::println*)*;  
*// 生成  
// public static<T> Stream<T> generate(Supplier<T> s)* Stream.*generate(*Math::*random)*.limit*(*10*)*.forEach*(*System.*out*::println*)*;  
 *}*

1. 步驟二：中間操作
2. 步驟三：終止操作





Collector需要使用Collectors提供實例。

