New Features of JDK

8

- http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html
- http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

Review of JDK7 New Features

- Binary Literals
- Strings in switch Statements
- The try-with-resources Statement
- Catching Multiple Exception Types and Rethrowing Exceptions with Improved Type Checking
- Underscores in Numeric Literals
- Type Inference for Generic Instance Creation e.g.Map<String, List<String>> myMap = new HashMap<>();
- Fork/Join Framework

Programming Language

- Lambda Expression
- Method Reference
- Default Method
- Repeating Annotation
- Annotation Anywhere
- Improved Type Inference
- Method Parameter Reflection

Collection

- Stream API
- HashMap with Key Collision

Concurrency

- New Class & Interface in java.util.concurrent
- Improved ConcurrentHashMap
- java.util.concurrent.atomic
- ForkJoinPool to Support Common Pool
- java.util.concurrent.lock.StampedLock

Date-Time Package

- a new set of packages that provide a comprehensive date-time model.

java.lang & java.util

- Parallel Array Sorting
- Standard Base64
- Unsigned Arithmetic Support

HotSpot

- Hardware based AES
- Removal of PermGen
- Support of Default Method in byte code instruction

JavaScript Engine

- Nashorn

Lambda Expression

- One issue with anonymous classes is that if the implementation of your anonymous class is very simple, such as an interface that contains only one method, then the syntax of anonymous classes may seem unwieldy and unclear. In these cases, you're usually trying to pass functionality as an argument to another method, such as what action should be taken when someone clicks a button. Lambda expressions enable you to do this, to treat functionality as method argument, or code as data.

Demo

LambdaExpressionDemo_FirstLook.java

Lambda Expression—Grammar

- A comma-separated list of formal parameters enclosed in parentheses. You can omit the data type of the parameters in a lambda expression. In addition, you can omit the parentheses if there is only one parameter.

- The arrow token, ->

- A body, which consists of a single expression or a statement block. If you specify a single expression, then the Java runtime evaluates the expression and then returns its value. Alternatively, you can use a return statement. A return statement is not an expression; in a lambda expression, you must enclose statements in braces ({}). However, you do not have to enclose a void method invocation in braces. Note that a lambda expression looks a lot like a method declaration; you can consider lambda expressions as anonymous methods—methods without a name.

Accessing Local Variables of the Enclosing Scope

Demo

LambdaExpression_Scope.java

Target Typing

- When the Java runtime invokes a method involving lambda expression, it's expecting a data type of the declaring class, so the lambda expression is of this type. The data type that these methods expect is called the target type. To determine the type of a lambda expression, the Java compiler uses the target type of the context or situation in which the lambda expression was found. It follows that you can only use lambda expressions in situations in which the Java compiler can determine a target type:

- Variable declarations
- Assignments
- Return statements
- Array initializers
- Method or constructor arguments
- Lambda expression bodies
- Conditional expressions, ?:
- Cast expressions

Target Types and Method Arguments

- For method arguments, the Java compiler determines the target type with two other language features: overload resolution and type argument inference.

Demo

LambdaExpression_Typing.java

Serialization

- You can serialize a lambda expression if its target type and its captured arguments are serializable. However, like inner classes, the serialization of lambda expressions is strongly discouraged.

Back to Menu

Method Reference

Demo

MethodReference.java

Kinds of Method References

Kind	Example
Reference to a static method	ContainingClass::staticMethodName
Reference to an instance method of a particular object	ContainingObject::instanceMethodName
Reference to an instance method of an arbitrary object of a particular type	ContainingType::methodName
Reference to a constructor	ClassName::new

Default Method

- Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.

Demo

Driver.java

Extending Interfaces That Contain Default Methods

- Not mention the default method at all, which lets your extended interface inherit the default method.
- Redeclare the default method, which makes it abstract.
- Redefine the default method, which overrides it.

Stream API

Demo

StreamAPI.java

Streamline

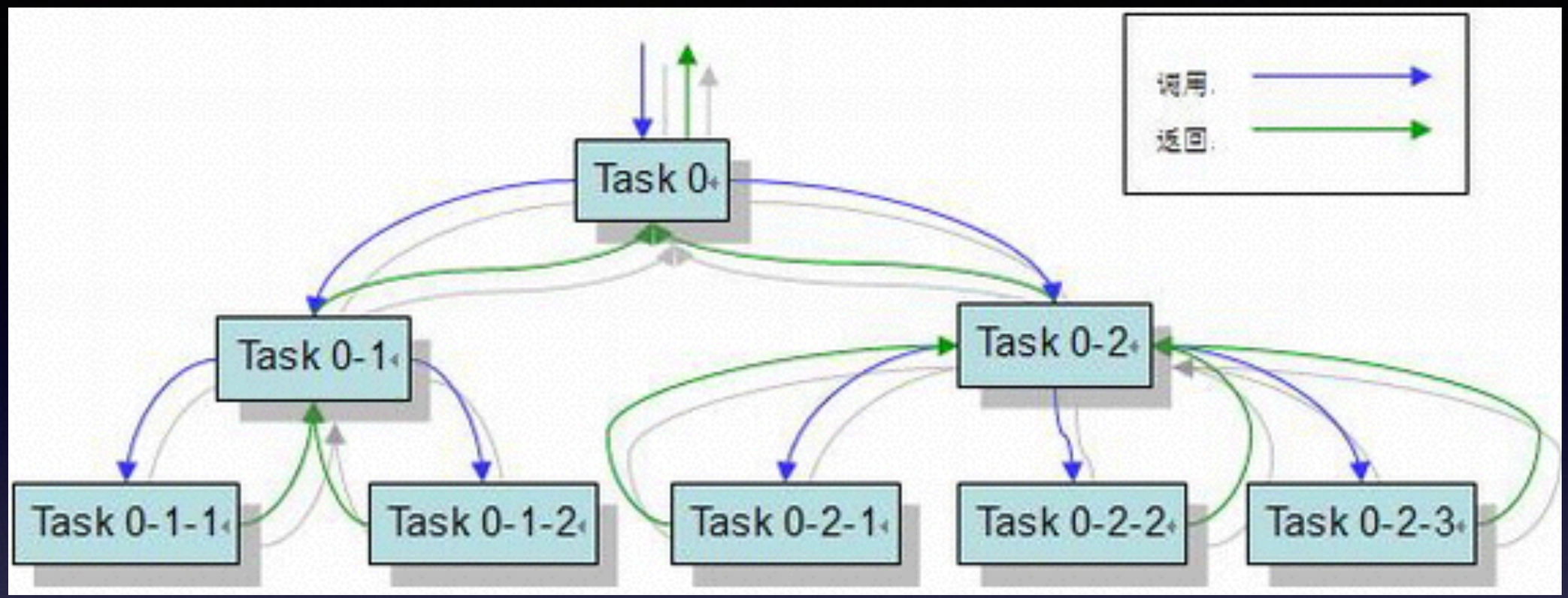
- A pipeline contains the following components:
- A source: This could be a collection, an array, a generator function, or an I/O channel.
- Zero or more intermediate operations. An intermediate operation, such as filter, produces a new stream.
  A stream is a sequence of elements. Unlike a collection, it is not a data structure that stores elements. Instead, a stream carries values from a source through a pipeline. Our example creates a stream from the collection roster by invoking the method stream.
  The filter operation returns a new stream that contains elements that match its predicate (this operation's parameter).
- A terminal operation. A terminal operation, such as forEach, produces a non-stream result, such as a primitive value (like a double value), a collection, or in the case of forEach, no value at all.

Fork/Join

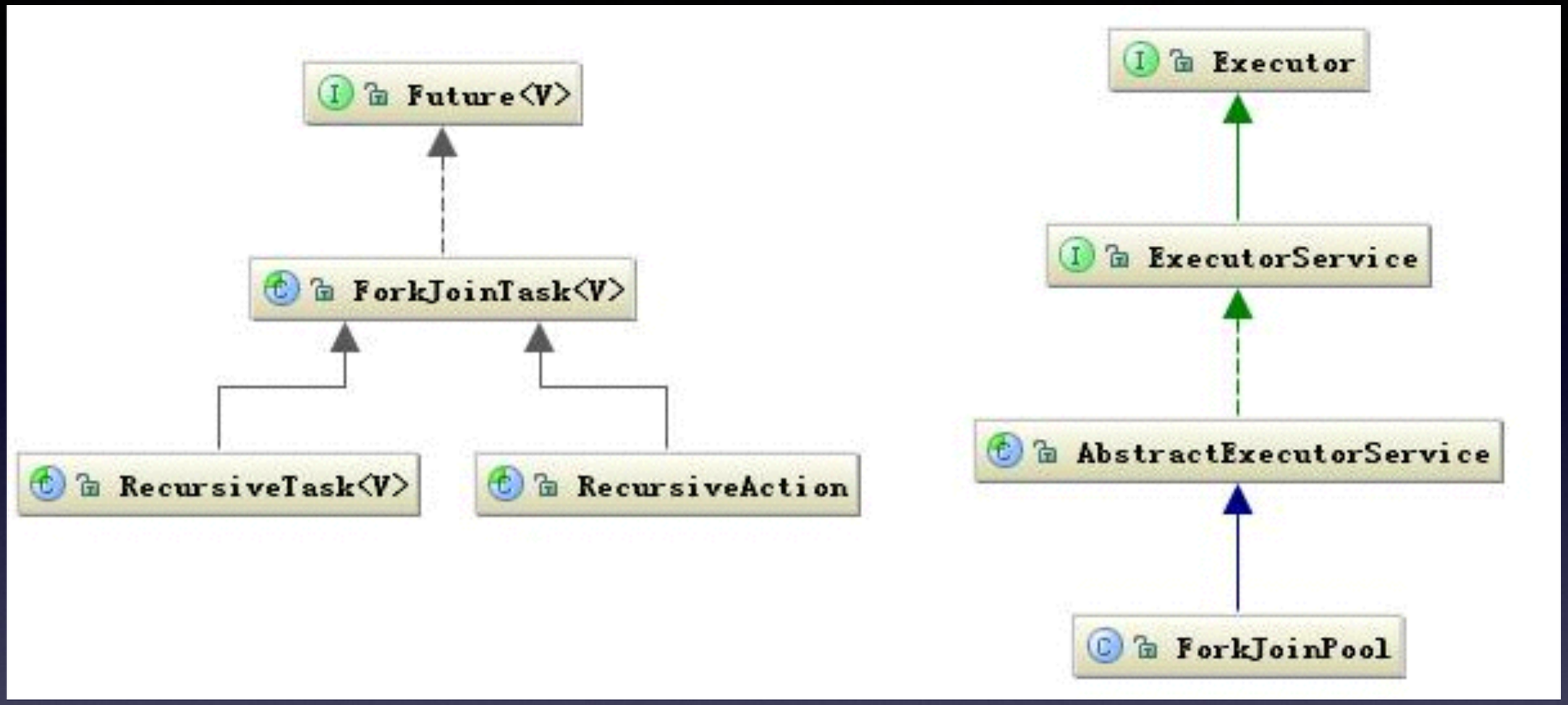
- The fork/join framework is an implementation of the ExecutorService interface that helps you take advantage of multiple processors. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application.

- As with any ExecutorService implementation, the fork/join framework distributes tasks to worker threads in a thread pool. The fork/join framework is distinct because it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy.

- The center of the fork/join framework is the ForkJoinPool class, an extension of the AbstractExecutorService class. ForkJoinPool implements the core work-stealing algorithm and can execute ForkJoinTask processes.



Thread Model


Future<V>
ForkJoinTask<V>
RecursiveTask<V>
RecursiveAction
Executor
ExecutorService
AbstractExecutorService
ForkJoinPool

Main Class

Demo

ForkJoinDemo.java

AES

- -XX:+UseAES -XX:+UseAESIntrinsics
- The hardware must be 2010 or newer Westmere hardware.