Better Java Builds with the Mill Build Tool

Li Haoyi
JJUG CCC 2024, Tokyo 27 Oct 2024

0.1 About Myself

Open Source Developer since 2012

- Projects with >10,000 stars on github

Prior Devtools at Dropbox, Databricks

Author of Hands-on Scala Programming

Better Java Builds with the Mill Build Tool

1. What is Mill?
2. Why Mill?
3. How does Mill work?

Better Java Builds with the Mill Build Tool

1. **What is Mill?**

2. Why Mill?

3. How does Mill work?

1. What is Mill?

Mill is an open-source JVM build tool, since 2017. Previously for Scala, now Java and Kotlin

Mostly does the same thing as Maven or Gradle

Aims to be easier to use and extend than Maven or Gradle

Currently 5-10x faster than Maven, 2-4x faster than Gradle for equivalent local builds

Documentation at mill-build.org

1.1 Simple Mill Build

```scala
// build.mill
package build
import mill._, javalib._

object foo extends JavaModule {
  def ivyDeps = Agg(
    ivy"net.sourceforge.argparse4j:argparse4j:0.9.0",
    ivy"org.thymeleaf:thymeleaf:3.1.1.RELEASE"
  )

  object test extends JavaTests with TestModule.Junit4
}
```

```
> /mill foo.compile
compiling 1 Java source...

> /mill foo.run --text hello
<h1>hello</h1>

> ./mill foo.test
Test foo.FooTest.testEscaping finished, ...
Test foo.FooTest.testSimple finished, ...
0 failed, 0 ignored, 2 total, ...

> ./mill show foo.assembly
".../out/foo/assembly.dest/out.jar"

> ./out/foo/assembly.dest/out.jar --text hello
<h1>hello</h1>
```

1.2 Mill Demo

Better Java Builds with the Mill Build Tool

1. What is Mill?

2. **Why Mill?**

3. How does Mill work?

2. Why Mill?

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

2.1 Why Mill? Performance

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

2.1.1 Mill vs Maven: Live Demo

Benchmark on the github.com/netty/netty codebase

~500,000 lines of code, ~2,900 Java files, ~50 subprojects

Comparing the existing build (Maven) with a custom Mill build. Tests pass in both

2.1.2 Mill vs Maven: Statistics

Benchmark	Maven	Mill	Speedup
Sequential Clean Compile All	2m 31.12s	0m 22.19s	6.8x
Parallel Clean Compile All	1m 16.45s	0m 09.95s	7.7x
Clean Compile Single-Module	0m 19.62s	0m 02.17s	9.0x
Incremental Compile Single-Module	0m 21.10s	0m 00.54s	39.1x
No-Op Compile Single-Module	0m 17.34s	0m 00.47s	36.9x

2.1.2 Mill vs Maven: Statistics

Benchmark	Maven	Mill	Speedup
Sequential Clean Compile All	2m 31.12s	0m 22.19s	6.8x
Parallel Clean Compile All	1m 16.45s	0m 09.95s	7.7x
Clean Compile Single-Module	0m 19.62s	0m 02.17s	9.0x
Incremental Compile Single-Module	0m 21.10s	0m 00.54s	39.1x
No-Op Compile Single-Module	0m 17.34s	0m 00.47s	36.9x

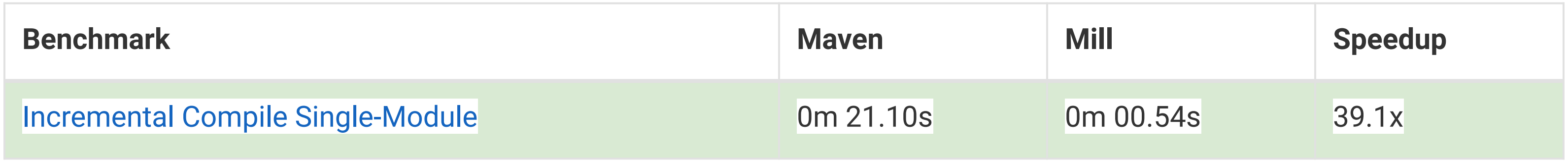
2.1.2 Mill vs Maven: Statistics

Benchmark	Maven	Mill	Speedup
Parallel Clean Compile All	1m 16.45s	0m 09.95s	7.7x

```
time ./mvnw -T10 -DskipTests -Dcheckstyle.skip -Denforcer.skip=true clean install
```

```
./mill clean && time ./mill __.compile
```

2.1.2 Mill vs Maven: Statistics



Benchmark	Maven	Mill	Speedup
Incremental Compile Single-Module	0m 21.10s	0m 00.54s	39.1x

```
echo "" >> common/src/main/java/io/netty/util/AbstractConstant.java

time ./mvnw -pl common -DskipTests  -Dcheckstyle.skip -Denforcer.skip=true install

time ./mill common.test.compile
```

2.1.2 Mill vs Maven: Statistics

Benchmark	Maven	Mill	Speedup
Sequential Clean Compile All	2m 31.12s	0m 22.19s	6.8x
Parallel Clean Compile All	1m 16.45s	0m 09.95s	7.7x
Clean Compile Single-Module	0m 19.62s	0m 02.17s	9.0x
Incremental Compile Single-Module	0m 21.10s	0m 00.54s	39.1x
No-Op Compile Single-Module	0m 17.34s	0m 00.47s	36.9x

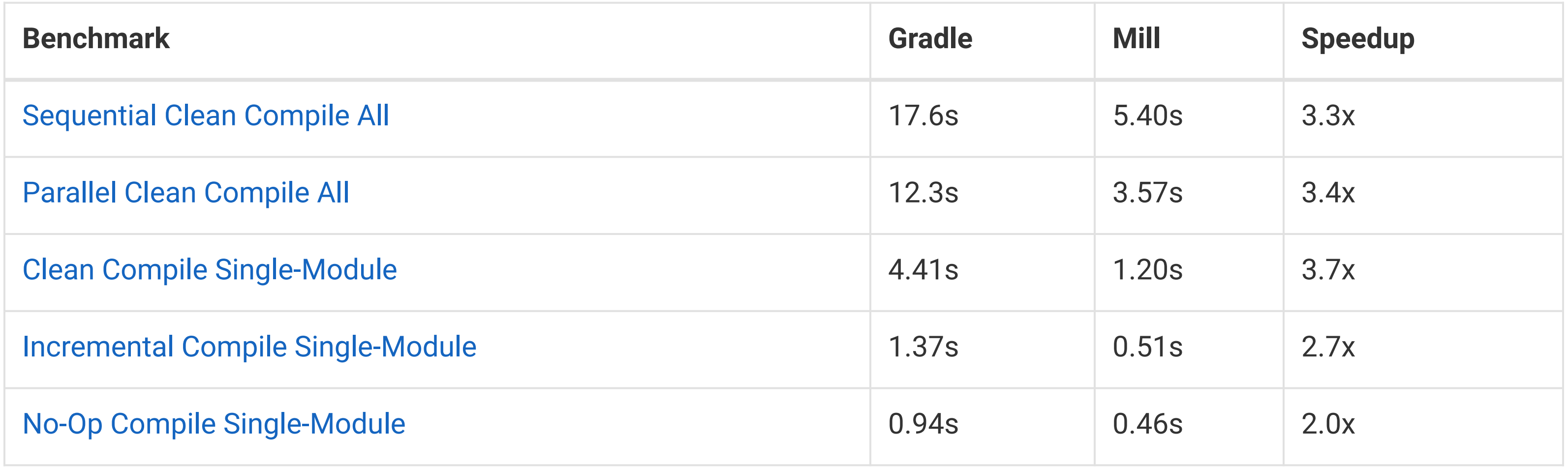
2.1.3 Mill vs Gradle: Live Demo

Benchmark on the github.com/mockito/mockito codebase

~100,000 lines of code, ~1,000 Java files, ~20 subprojects

Comparing the existing build (Gradle) with a custom Mill build. Tests pass in both

2.2.3 Mill vs Gradle: Statistics

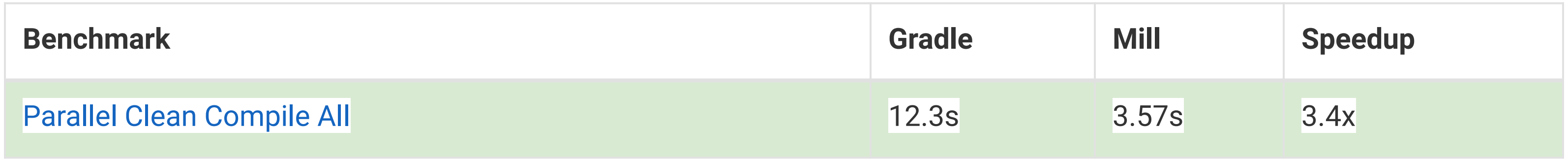


Benchmark	Gradle	Mill	Speedup
Sequential Clean Compile All	17.6s	5.40s	3.3x
Parallel Clean Compile All	12.3s	3.57s	3.4x
Clean Compile Single-Module	4.41s	1.20s	3.7x
Incremental Compile Single-Module	1.37s	0.51s	2.7x
No-Op Compile Single-Module	0.94s	0.46s	2.0x

2.2.3 Mill vs Gradle: Statistics

Benchmark	Gradle	Mill	Speedup
Sequential Clean Compile All	17.6s	5.40s	3.3x
Parallel Clean Compile All	12.3s	3.57s	3.4x
Clean Compile Single-Module	4.41s	1.20s	3.7x
Incremental Compile Single-Module	1.37s	0.51s	2.7x
No-Op Compile Single-Module	0.94s	0.46s	2.0x

2.2.3 Mill vs Gradle: Statistics



Benchmark	Gradle	Mill	Speedup
Parallel Clean Compile All	12.3s	3.57s	3.4x

```
./gradlew clean; time ./gradlew classes testClasses --no-build-cache

./mill clean; time ./mill __.compile
```

2.2.3 Mill vs Gradle: Statistics

Benchmark	Gradle	Mill	Speedup
Incremental Compile Single-Module	1.37s	0.51s	2.7x

```
echo "" >> src/main/java/org/mockito/BDDMockito.java; time ./gradlew :classes

echo "" >> src/main/java/org/mockito/BDDMockito.java; time ./mill compile
```

2.2.3 Mill vs Gradle: Statistics

Benchmark	Gradle	Mill	Speedup
Sequential Clean Compile All	17.6s	5.40s	3.3x
Parallel Clean Compile All	12.3s	3.57s	3.4x
Clean Compile Single-Module	4.41s	1.20s	3.7x
Incremental Compile Single-Module	1.37s	0.51s	2.7x
No-Op Compile Single-Module	0.94s	0.46s	2.0x

2.1 Why Mill? Performance

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly import any Java library from Maven Central to use in your build

2.2 Why Mill? Ease of Use

1. Performance

- 2-10x speedup means less time waiting for your build tool

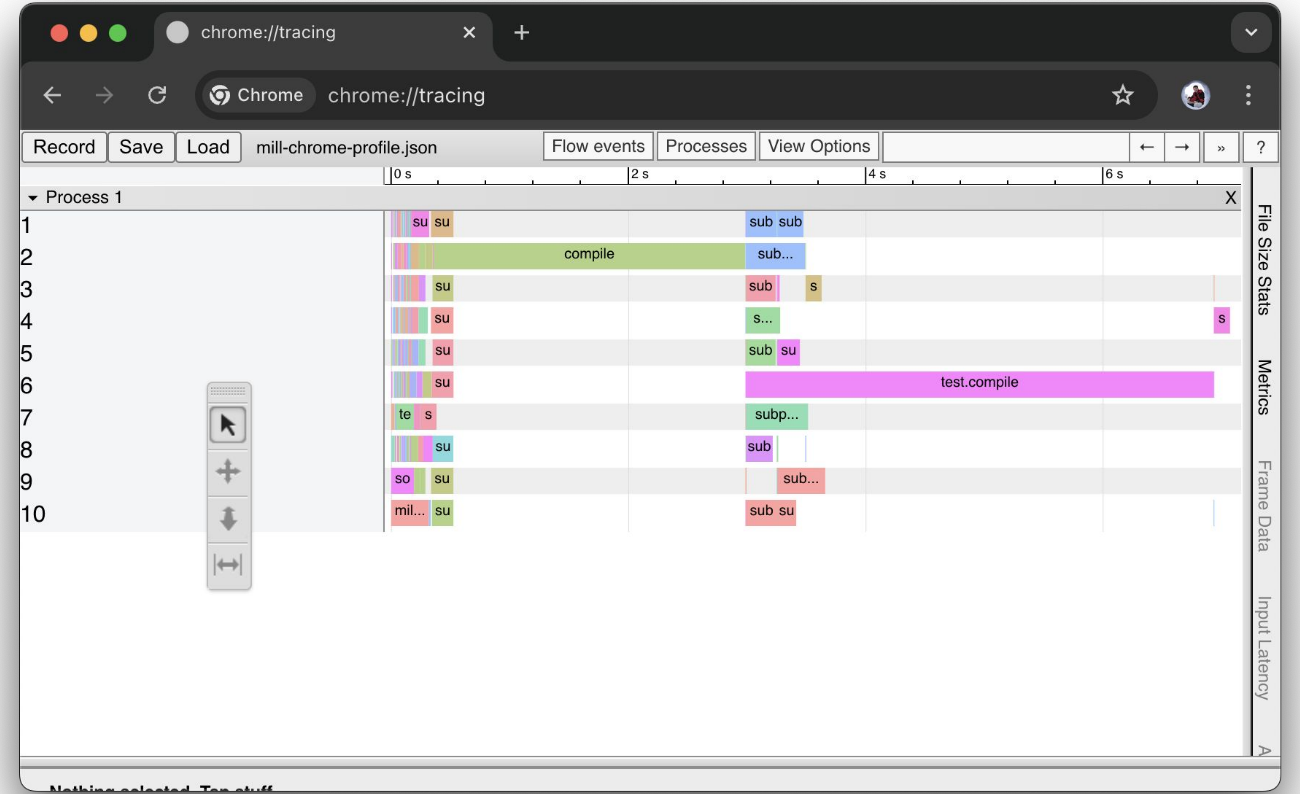
<u>2. Ease of Use</u>

- Full IDE support in IntelliJ and VSCode, introspection tools

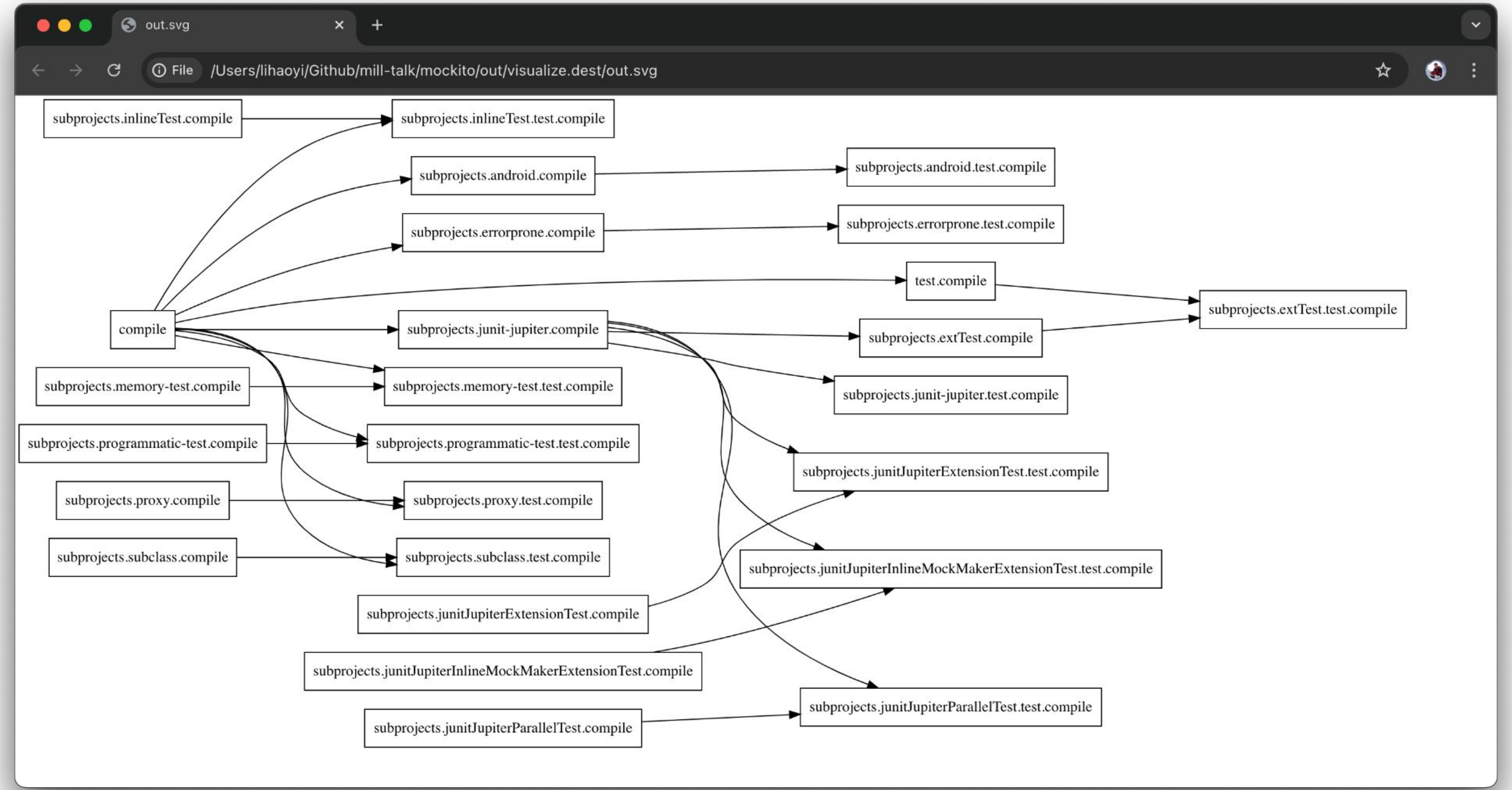
3. Extensibility

- Directly import any library from Maven Central to use in your build

2.2.3 Mill Chrome Profile

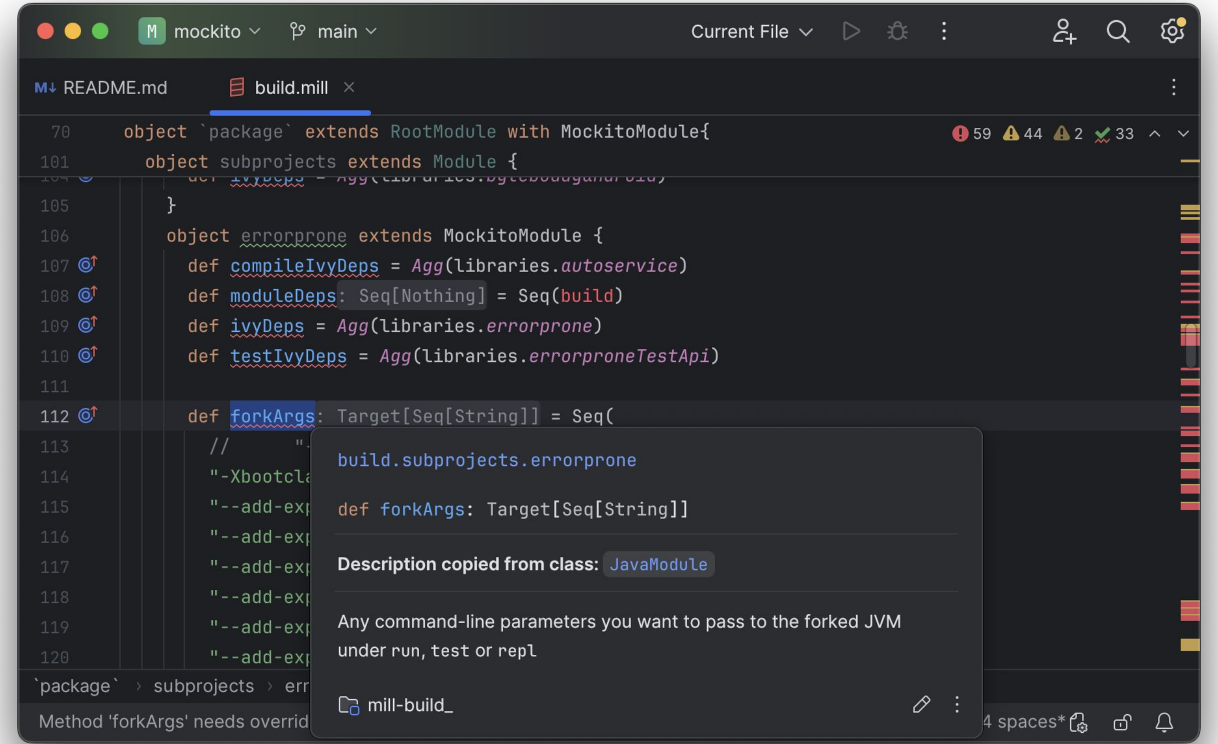


2.2.3 Mill Visualize



out.svg
File
/Users/lihaoyi/Github/mill-talk/mockito/out/visualize.dest/out.svg
subprojects.inlineTest.compile
subprojects.inlineTest.test.compile
subprojects.android.compile
subprojects.android.test.compile
subprojects.errorprone.compile
subprojects.errorprone.test.compile
test.compile
compile
subprojects.junit-jupiter.compile
subprojects.extTest.test.compile
subprojects.extTest.compile
subprojects.memory-test.compile
subprojects.memory-test.test.compile
subprojects.junit-jupiter.test.compile
subprojects.programmatic-test.compile
subprojects.programmatic-test.test.compile
subprojects.junitJupiterExtensionTest.test.compile
subprojects.proxy.compile
subprojects.proxy.test.compile
subprojects.subclass.compile
subprojects.subclass.test.compile
subprojects.junitJupiterInlineMockMakerExtensionTest.test.compile
subprojects.junitJupiterExtensionTest.compile
subprojects.junitJupiterInlineMockMakerExtensionTest.compile
subprojects.junitJupiterParallelTest.test.compile
subprojects.junitJupiterParallelTest.compile

2.2.2 IDE support demo: Mill vs Gradle



```
object `package` extends RootModule with MockitoModule{
  object subprojects extends Module {
    }
    object errorprone extends MockitoModule {
      def compileIvyDeps = Agg(libraries.autoservice)
      def moduleDeps: Seq[Nothing] = Seq(build)
      def ivyDeps = Agg(libraries.errorprone)
      def testIvyDeps = Agg(libraries.errorproneTestApi)

      def forkArgs: Target[Seq[String]] = Seq(
        //      ".
        "-Xbootcla
        "--add-exp
        "--add-exp
        "--add-exp
        "--add-exp
        "--add-exp
        "--add-exp
```

build.subprojects.errorprone

`def forkArgs: Target[Seq[String]]`

Description copied from class: `JavaModule`

Any command-line parameters you want to pass to the forked JVM under `run`, `test` or `repl`

mill-build_

`package` › subprojects › err

Method 'forkArgs' needs overrid

4 spaces*

2.2 Why Mill? Ease of Use

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Introspection tools, great IDE support in IntelliJ and VSCode,

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

2.3 Why Mill? Extensibility

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

2.3.1 Plugin-Free Extensibility

Most build tools require extensions to be published as “plugins”

Mill lets you to *write code* or *import any library from Maven Central* to use!

2.3.2 Simple Overrides

```scala
package build
import mill._, javalib._

object foo extends JavaModule {
}
```

```
> mill compile

Compiling 1 Java source...
```

2.3.2 Simple Overrides

```scala
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }
}
```

```
> mill show foo.lineCount

17
```

2.3.2 Simple Overrides

```scala
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

```
> mill show foo.lineCount

17
```

```
> mill foo.run

Line Count: 17
```

2.3.3 import $ivy

```scala
package build
import mill._, javalib._
import $ivy.`org.thymeleaf:thymeleaf:3.1.1.RELEASE`
import org.thymeleaf.TemplateEngine
import org.thymeleaf.context.Context
object foo extends JavaModule {
  def htmlSnippet = Task {
    val context = new Context()
    context.setVariable("heading", "hello")
    new TemplateEngine().process(
        "<h1 th:text=\"${heading}\"></h1>",
        context
    )
  }
  def resources = Task.Sources{
    os.write(Task.dest / "snippet.txt", htmlSnippet())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

```
> mill foo.compile

compiling 1 Java source...

...
```

```
> mill foo.run

generated snippet.txt resource:

<h1>hello</h1>
```

2.3. Why Mill? Extensibility

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

2. Why Mill?

1. Performance

- 2-10x speedup means less time waiting for your build tool

2. Ease of Use

- Full IDE support in IntelliJ and VSCode, introspection tools

3. Extensibility

- Directly write code or use any Java library from Maven Central in your build

Better Java Builds with the Mill Build Tool

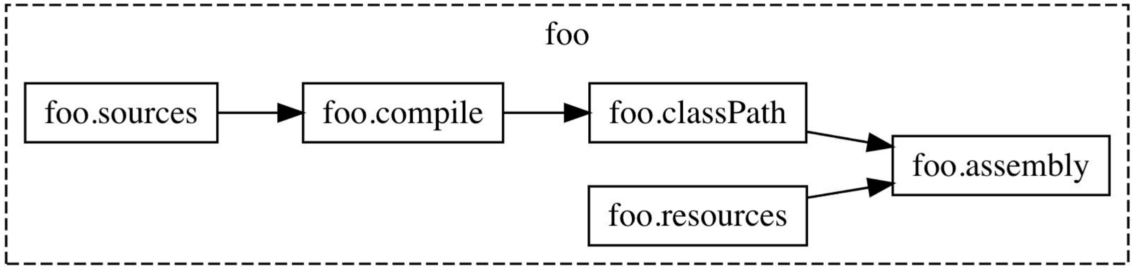
1. What is Mill?

2. Why Mill?

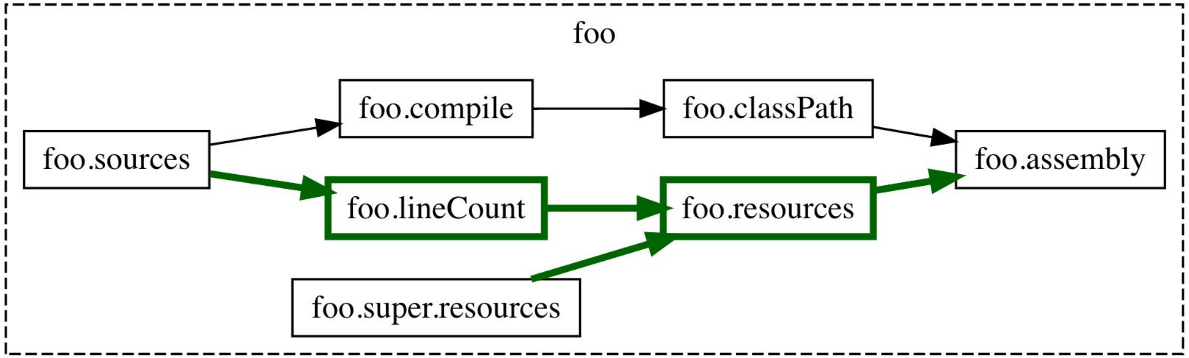
3. **How does Mill work?**

3.1 How Mill Works

```
package build
import mill._, javalib._

object foo extends JavaModule {
}
```

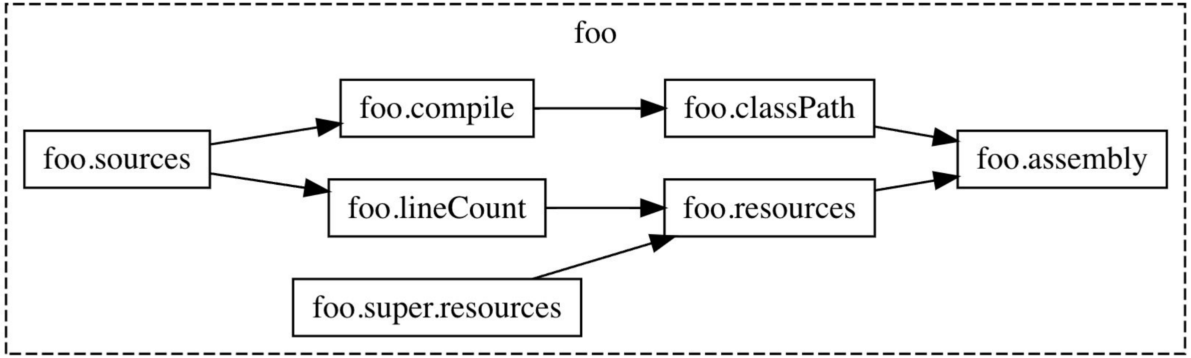


3.2 How Mill Works



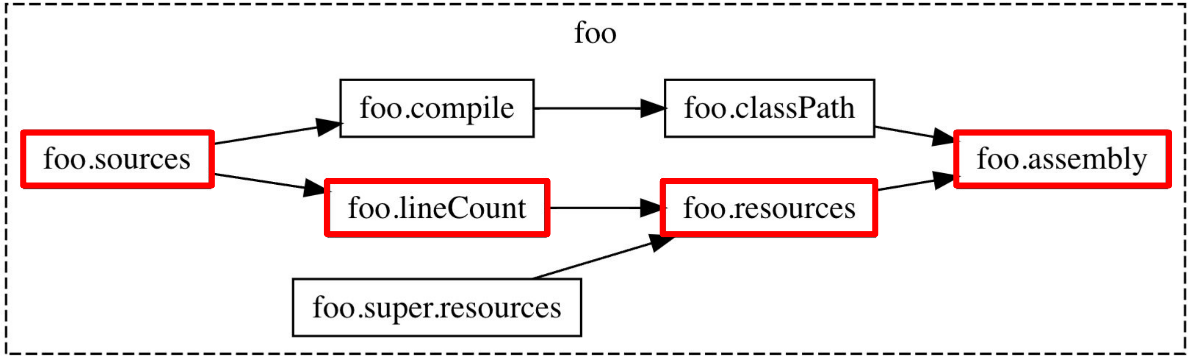
```
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

3.3 How Mill Works



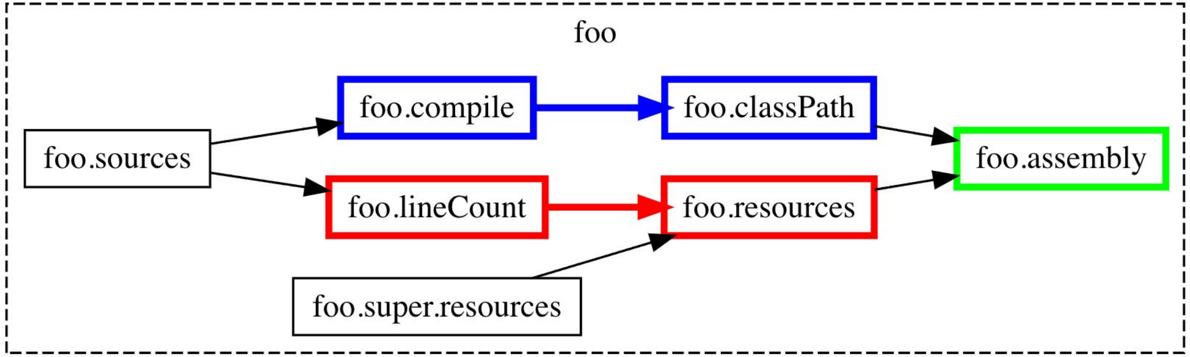
```
object foo extends JavaModule {
  def sources = Task { ... }

  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }

  def assembly = Task { ... }
}
```

3.4 How Mill Works



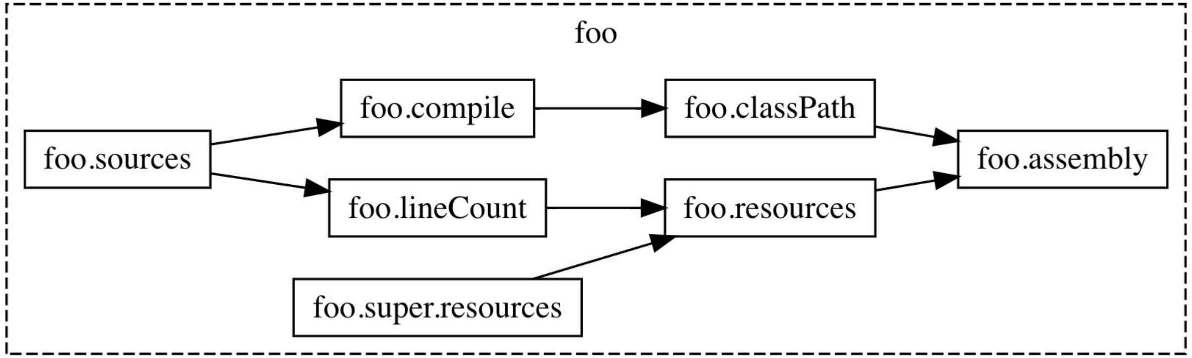
```
object foo extends JavaModule {
  def sources = Task { ... }

  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }

  def assembly = Task { ... }
}
```

3.5 How Mill Works



```
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

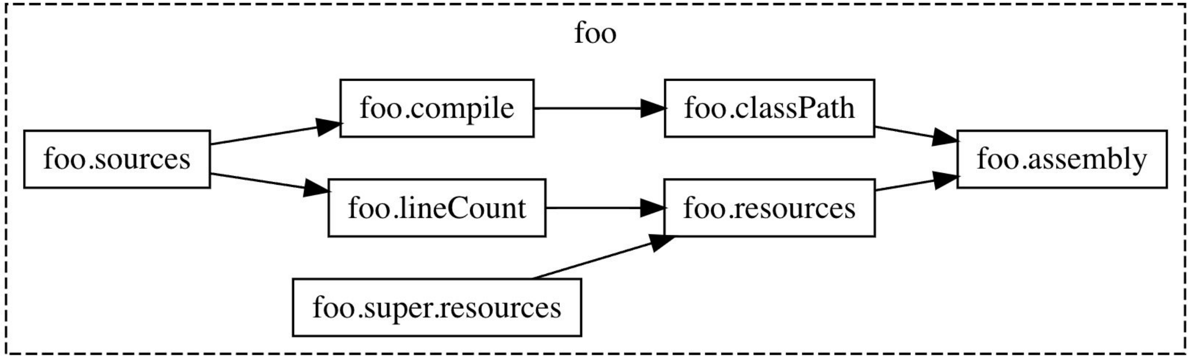
3.6 How Mill Works



```
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

```
foo.lineCount
    out/foo/lineCount.json

foo.compile
    out/foo/compile.json
    out/foo/compile.dest/
```

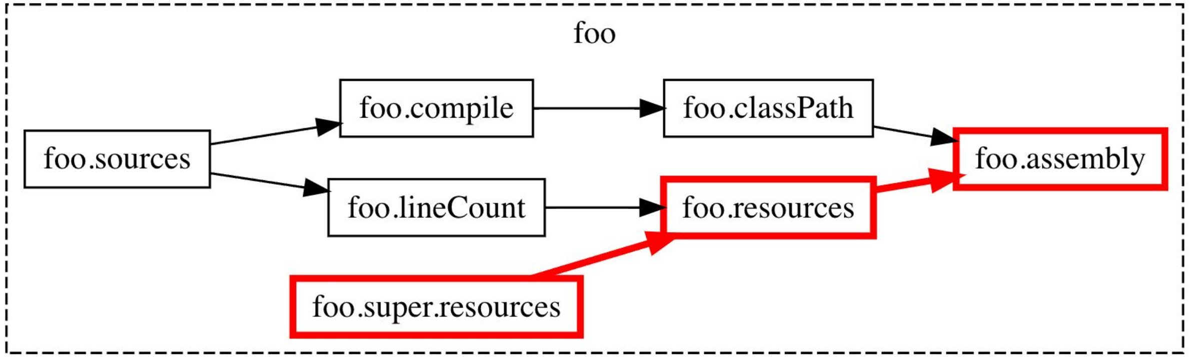
3.6 How Mill Works



```
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

```
foo.lineCount
    out/foo/lineCount.json

foo.compile
    out/foo/compile.json
    out/foo/compile.dest/

foo.assembly
    out/foo/assembly.json
    out/foo/assembly.dest/
```

3.7 How Mill Works



```scala
package build
import mill._, javalib._

object foo extends JavaModule {
  /** Total number of lines in module source files */
  def lineCount = Task {
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  /** Generate resources using lineCount of sources */
  override def resources = Task {
    os.write(Task.dest / "line-count.txt", "" + lineCount())
    super.resources() ++ Seq(PathRef(Task.dest))
  }
}
```

```
foo.lineCount
    out/foo/lineCount.json

foo.compile
    out/foo/compile.json
    out/foo/compile.dest/

foo.assembly
    out/foo/assembly.json
    out/foo/assembly.dest/
```

Better Java Builds with the Mill Build Tool

1. What is Mill?

2. Why Mill?

3. **How does Mill work?**

Better Java Builds with the Mill Build Tool

1. What is Mill?
   a. Mill is an open-source JVM build tool for Java, Scala, Kotlin

2. Why Mill?
   a. Performance: 5-10x faster than Maven, 2-4x faster than Gradle
   b. Ease of Use: good visualization tools and IDE support in IntelliJ and VSCode
   c. Extensibility: Directly write code or import any library from Maven Central to use in your build

3. How does Mill work?
   a. Modules are just objects, Tasks are just methods
   b. Module Tree and Task Graph, automatic caching and parallelization

4. mill-build.org