JDK9 新特性

- **目录结构**
- **模块化系统**
- **jshell**
- **接口的私有方法**
- **改进try-with-resourcs**
- **改进砖石操作符**
- **限制使用单独下划线标识符**
- **String存储结构变更**
- **快速创建只读结合**
- **增强Stream API**
- **改进Optional 类**
- **多分辨率图像 API**
- **全新 HTTP客服端API**
- **智能JAVA 编译工具**
- **统一JVM 日志系统**
- **javadoc 的 HTML5 支持**
- **java 动态编译**

一、目录结构

JDK9具体目录结构如下所示：

- bin： 该目录包含所有的命令。
- conf： 包含用户可以编辑的配置文件，例如以前位于jre\lib 目录中的.properties 和 .policy 文件。
- include： 包含一些编译本地代码时使用的C/C++头文件。
- jmods： 包含JMOD 格式的平台模块，创建自定义运行映射时需要它。
- legal： 包含法律声明。
- lib： 包含非Windows 平台上动态链接的本地库，其子目录和文件不应由开发人员直接编辑或使用。

注：JDK9 目录中不再有jre子目录。

二、模块化系统

JDK9将JDK分成一组模块，可以在编译时，运行时或构建时进行组合。模块化可以减少内存开销；只需必要的模块，并非全部模块，可以简化各种类库和大型应用的开发和维护，八个仓库：root、corba、hotspot、jaxp、jaxws、jdk、langtools和nashorn

module-info.java：该文件必须位于项目的根目录中。该文件用于定义模块需要什么依赖，以及那些包被外部使用。
exports：控制着那些包可以被其他模块访问到，所有不被exports的包默认都被封装在模块里面不被外界所使用。
requires：指明对其他模块的依赖。

三、JSHELL

JDK9新增了REPL（Read-Eval-Print Loop）工具jshell，jshell工具提供了一个交互式命令界面，可以评估声明，语句和表达式，无需编译即可返回执行结果。无论是在初学JAVA或学习新的API时都非常有用

```
/list //列出所有的代码
/methods //查看所有的方法
/var //所有的变量
/edit //打开编辑器
/open path //执行路径上的代码 如 /open C:\Users\wukong\Desktop\App.java
```

四、接口的私有化方法

在JDK8中接口运行使用静态方法和默认方法后，JDK9可以在接口中使用私有方法

```
@FunctionalInterface
public interface TestFunctionalInterface {
 //todo @FunctionalInterface注解会检测接口是否有且只有一个抽象方法，还可以有默认方法和静态方法
  public void test();
  //todo 默认方法
  default String getName(){
  log("getName");
  return "人参";
  }
   //todo 默认方法
   default String getDesc(){
   log("getDesc");
   return "人参大补元气";
   }
   private void log(String method){
```

```
 System.out.println("TestFunctionalInterface."+method+"()被调用");
 }
 //todo 静态方法
 static String getName2(){
 return "鹿茸";
 }
}
```

五、改进try-with-resource

JDK8中新增了try-with-resources语句，可以自动关闭需要关闭的资源文件。但是必须在try语句后的括号中初始化需要关闭的资源。在JDK9中改进了try-with-resources语句，你可以在try外初始化资源，然后在try后的括号中添加需要自动关的资源即可

```
public static void jdk7() {
 FileInputStream fileInputStream = null;

 try {
 fileInputStream = new FileInputStream("D:/a.txt");
 byte[] temp = new byte[1024];
 fileInputStream.read(temp);
 System.out.println(new String(temp));
 } catch (Exception e) {
 e.printStackTrace();
 }finally {
 if (fileInputStream != null){
 try {
 fileInputStream.close();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
 }
}
public static void jdk8() {


 try( FileInputStream fileInputStream = new FileInputStream("D:/a.txt"))
{

 byte[] temp = new byte[1024];
 fileInputStream.read(temp);
```

```
 System.out.println(new String(temp));
 } catch (Exception e) {
 e.printStackTrace();
 }
}
public static void jdk9() throws FileNotFoundException {

 FileInputStream fileInputStream = new FileInputStream("D:/a.txt");
 try( fileInputStream) {
 byte[] temp = new byte[1024];
 fileInputStream.read(temp);
 System.out.println(new String(temp));
 } catch (Exception e) {
 e.printStackTrace();
 }
}
```

六、改进钻石操作符

JDK9中钻石操作符可以使用匿名实现类，可以在匿名实现类中重写方法等操作

```
public static void main(String[] args) {
 //jdk9允许
 Comparator<Integer> comparator = new Comparator<>() {
 @Override
 public int compare(Integer o1, Integer o2) {
 return 0;
 }
 };
}
```

七、限制使用单独下划线标识符

在JDK8之前可以使用“_”单独的下划线作为标识符，但在JDK9中将单独的下划线标识符限制使用了

```
public static void main(String[] args) {
 int _ = 20;//jdk9已禁止
 System.out.println(_);
}
```

八、String存储结构变更

从很多不同应用程序收集的信息表名，字符串是堆使用的主要组成部分，而且，大多数字符串对象只包含一个字符，这样的字符只需要一个字节的存储空间，因此这些字符串对象的内部char数组中

有一半的空间被闲置。

JDK9之前String底层使用char数组存储数据private final char value[]，JDK9将String底层存储数据改为byte数组存储数据private final byte[] value。

StringBuffer和StringBuilder也同样做了变更，将以往char数组改为byte数组。

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {

    /**
     * The value is used for character storage.
     *
     * @implNote This field is trusted by the VM, and is a subject to
     * constant folding if String instance is constant. Overwriting this
     * field after construction will cause problems.
     *
     * Additionally, it is marked with {@link Stable} to trust the contents
     * of the array. No other facility in JDK provides this functionality (yet).
     * {@link Stable} is safe here, because value is never null.
     */
    @Stable
    private final byte[] value;
```

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
```

九、快速创建只读集合

JDK9在List、Set和Map集合中新增of静态方法，快速创建只读集合

```
List<String> list = List.of("三七","人参","鹿茸","紫河车");
list.add("黄连");//会抛出异常
Set<String> set = Set.of("三七","人参","鹿茸","紫河车");

Map<Integer,String> map = Map.of(1,"三七",2,"人参",3,"鹿茸",4,"紫河车");
```

十、增强Stream API

JDK9在Stream接口中新增4个方法：dropWhile、takeWhile、ofNullable,为iterate方法新增重载方法

```
Stream<String> stream = Stream.of("三七","人参","鹿茸","紫河车");
stream.takeWhile(item->item.length()==2).forEach(System.out::println);
System.out.println("================================================");
Stream<String> stream1 = Stream.of("三七","人参","鹿茸","紫河车");
stream1.dropWhile(item->item.length()==2).forEach(System.out::println);
System.out.println("================================================");
```

```
Stream<Integer> stream2 = Stream.of(7,40,6,333,4,4,5,2,3,3,3,3,6);
stream2.takeWhile(num->num<100).forEach(System.out::println);
System.out.println("=============================================");
 Stream<Integer> stream3 = Stream.of(7,40,6,333,4,4,5,2,3,3,3,3,6);
 stream3.dropWhile(num->num<100).forEach(System.out::println);
 System.out.println("=============================================");
 Stream<Integer> stream4 = Stream.ofNullable(null);
 System.out.println(stream4.count());
```

十一、改进Optional类

Optional 类是在JDK8中新增的类，主要是为了解决空指针异常。在JDK9中对这个类进行了改进，主要是新增了三个方法：stream，ifPresentOrElse 和 or

```
List<String> list = new ArrayList<>();
list.add("人参");
list.add("当归");
list.add("鹿茸");
list.add("黄柏");
Optional<List<String>> optional = Optional.ofNullable(list);
optional.stream().forEach(System.out::println);


 Optional<String> optional1 = Optional.of("Mahesh");
 optional1.ifPresentOrElse( x -> System.out.println("Value: " + x),() ->
  System.out.println("Not Present."));
 Supplier<Optional<String>> supplierString = () -> Optional.of("Not Present");
 optional1 = optional1.or( supplierString);
```

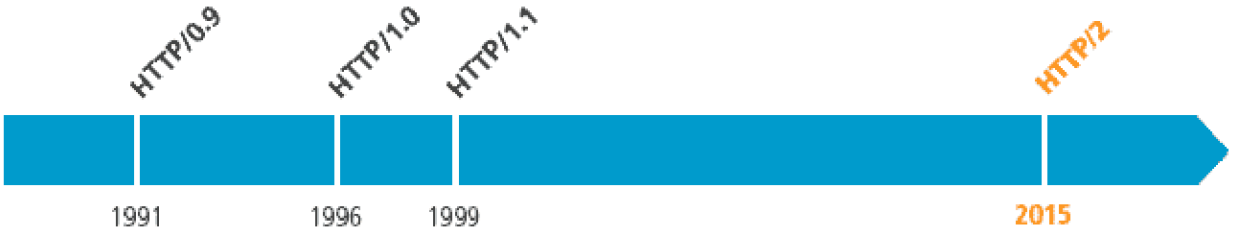
十二、多分辨率图像API

在 java.awt.image 包下新增了支持多分辨率图片的API，用于支持多分辨率的图片。

1. 将不同分辨率的图像封装到一张（多分辨率的）图像中，作为它的变体。
2. 获取这个图像的所有变体。
3. 获取特定分辨率的图像变体，表示一张已知分辨率单位为 DPI 的特定尺寸大小的逻辑图像，并且这张图像是最佳的变体。
4. java.awt.image.MultiResolutionImage接口的基础实现java.awt.image.BaseMultiResolutionImage获取所需要的变体。
5. 通过接口的getResolutionVariant（double destImageWidth, double destImageHeight）方法，根据分辨率获取图像。

十三、全新的HTTP客户端API

HTTP，用于传输网页的协议，早在 1997 年就被采用在目前的 1.1版本中。直到 2015 年，HTTP2 才成为标准。



HTTP/1.1和HTTP/2的主要区别是如何在客户端和服务器之间构建和传输数据。HTTP/1.1 依赖于请求/响应周期。 HTTP/2 允许服务器“push”数据：它可以发送比客户端请求更多的数据。 这使得它可以优先处理并发送对于首先加载网页至关重要的数据。

JDK9 中有新的方式来处理 HTTP 调用。它提供了一个新的HTTP客户端(HttpClient)，它将替代仅适用于blocking模式的HttpURLConnection(HttpURLConnection是在HTTP 1.0的时代创建的，并使用了协议无关的方法)，并提供对 WebSocket 和 HTTP/2 的支持。

此外，HTTP客户端还提供 API 来处理 HTTP/2 的特性，比如流和服务器推送等功能。

全新的 HTTP 客户端 API 可以从 jdk.incubator.httpclient 模块中获取。因为在默认情况下，这个模块是不能根据 classpath 获取的，需要使用 add modules 命令选项配置这个模块，将这个模块添加到 classpath中。

```
public static void main(String[] args) {
  //创建 builder
  HttpClient.Builder builder = HttpClient.newBuilder();
  //链式调用
  HttpClient client = builder
  //http 协议版本 1.1 或者 2
  .version(HttpClient.Version.HTTP_2) //.version(HttpClient.Version.HTTP_1_1)
  //连接完成之后的转发策略
  .followRedirects(HttpClient.Redirect.NEVER) //.followRedirects(HttpClient.Redirect.ALWAYS)
   //指定线程池
   .executor(Executors.newFixedThreadPool(5))

  //认证，默认情况下 Authenticator.getDefault() 是 null 值，会报错
  //.authenticator(Authenticator.getDefault())

  //代理地址
  //.proxy(ProxySelector.of(new InetSocketAddress("http://www.baidu.com", 8080)))

  //缓存，默认情况下 CookieHandler.getDefault() 是 null 值，会报错
  //.cookieHandler(CookieHandler.getDefault())

  //创建完成
```

```
23    .build();
24    //组装request
25    URI baiduUri = new URI("https://www.baidu.com");
26    HttpRequest request = HttpRequest.newBuilder(baiduUri)
27    .version(HttpClient.Version.HTTP_2)
28    .build();
29    //发起调用
30    HttpResponse<String> r = client.send(request,
HttpResponse.BodyHandler.asString());
31    System.out.println(r.body());
32  }
```

十四、智能JAVA编译工具

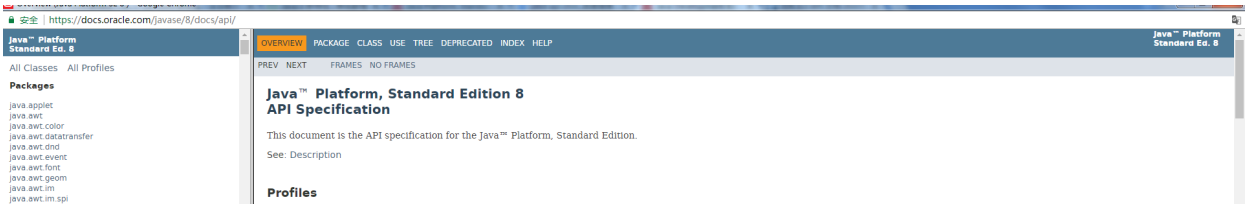
智能 java 编译工具(sjavac)的第一个阶段始于 JEP139 这个项目，用于在多核处理器情况下提升 JDK 的编译速度。如今，这个项目已经进入第二阶段，即 JEP199，其目的是改进 Java 编译工具，并取代目前 JDK 编译工具 javac，继而成为 Java 环境默认的通用的智能编译工具。
JDK 9 还更新了 javac 编译器以便能够将 java 9 代码编译运行在低版本 Java 中。

十五、统一的JVM日志系统

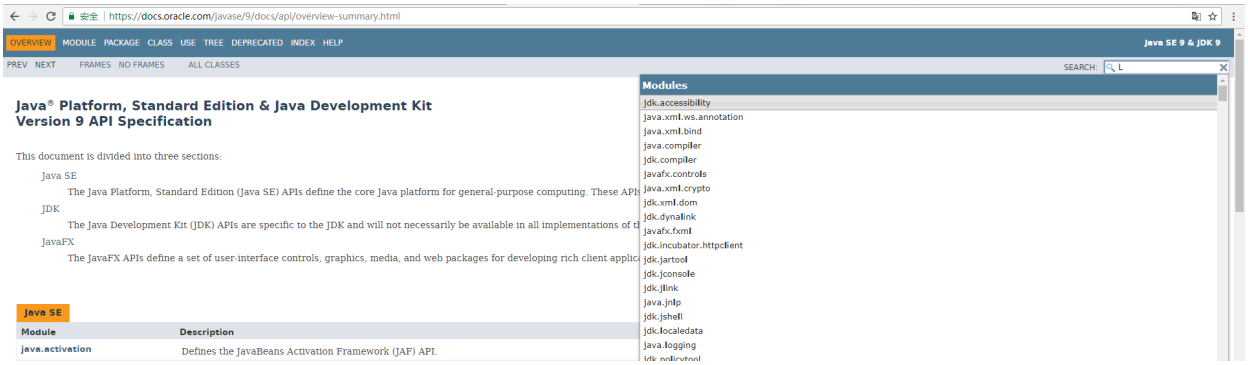
日志是解决问题的唯一有效途径：曾经很难知道导致 JVM 性能问题和导致 JVM 崩溃的根本原因。不同的 JVM 日志的碎片化和日志选项（例如：JVM 组件对于日志使用的是不同的机制和规则），这使得 JVM 难以进行调试。
解决该问题最佳方法：对所有的 JVM 组件引入一个单一的系统，这些 JVM 组件支持细粒度的和易配置的 JVM 日志。

十六、javadoc的HTML5支持

JDK8 生成的java帮助文档是在 HTML4 中。而HTML4 已经是很久的标准了。
JDK9 的javadoc，现支持HTML5 标准。
下图是JDK8 API的HTML页面



如下图是JDK9 API的 HTML，右上角支持搜索功能。



十七、java动态编译器

JIT（Just-in-time）编译器可以在运行时将热点编译成本地代码，速度很快。但是 Java 项目现在变得很大很复杂，因此 JIT 编译器需要花费较长时间才能热身完，而且有些 Java 方法还没法编译，性能方面也会下降。AoT 编译就是为了解决这些问题而生的。

在 JDK 9 中， AOT（JEP 295: Ahead-of-Time Compilation）作为实验特性被引入进来，开发者可以利用新的 jaotc 工具将重点代码转换成类似类库一样的文件。虽然仍处于试验阶段，但这个功能使得 Java应用在被虚拟机启动之前能够先将 Java 类编译为原生代码。此功能旨在改进小型和大型应用程序的启动时间，同时对峰值性能的影响很小。

但是 Java 技术供应商 Excelsior 的营销总监 Dmitry Leskov 担心 AoT 编译技术不够成熟，希望 Oracle 能够等到 Java 10时有个更稳定版本才发布。