大连理工大学本科毕业设计（论文）

代码级能耗分析工具设计与实现

Design and Implementation of Code Level Software Energy Consumption Analysis Tool

学 院（系）：软件学院

专　　业：软件工程

学 生 姓 名：唐智强

学　　号：201692191

指 导 教 师：侯刚

评 阅 教 师：王洁

完 成 日 期：2020 年 6 月 2 日

大连理工大学

Dalian University of Technology

摘　要

绿色 IT 已经发展成为了国际上专注于研究能源消耗以及设计优化的解决方案的一门重要学科。通过分析代码特征与代码能耗之间的关系，建立基于代码特征的软件能耗模型是软件能耗研究领域一个合理有效的研究思路，本文提取了软件代码的五大特征：有效代码行数、构件个数、算法复杂度、调用深度、耦合程度，并假设这些代码特征与软件能耗存在线性或非线性函数关系，在此基础上本文建立了一个代码静态能耗分析的预测模型。具体来说，通过两层架构，包括能耗评估模型和能耗预测模型，其中能耗评估模型从硬件及代码特征角度动态地监测进程的能耗，能耗预测模型从静态分析方面预测能耗。

该模型通过能耗评估模型进行具体测量，并使用能耗预测模型的神经网络拟合代码特征和能源消耗之间的线性或非线性函数关系。通过大量的实验结果验证，线性回归关系预测模型对特定类型的能耗值模型平均预测值在 10%以内，对非特定类型的预测值误差普遍高于 20%，非线性回归函数关系预测模型的非特定函数关系类型的能耗值模型平均预测值与实际应用中能耗值的预测误差均在 20%以内。这说明了两种模型的预测有效性和假设的差异性，即两种模型假设在特定的环境下都是合理有效的。本文还将对工具的具体实现进行描述。

关键词：绿色 IT ；能耗分析；能耗预测

Design and Implementation of Code Level Software Energy Consumption Analysis Tool

Abstract

Green IT has developed into an important discipline focusing on energy consumption and design optimization solutions internationally. By analyzing the relationship between code features and code energy consumption, it is a reasonable and effective research idea to build a software energy consumption model based on code features. In this paper, five characteristics of software code are extracted: the number of effective code lines, the number of components, the complexity of algorithm, the depth of call, the degree of coupling, and the linear relationship between these code features and software energy consumption is assumed On this basis, a prediction model of code static energy consumption analysis is established. Specifically, through two-tier architecture, including energy consumption evaluation model and energy consumption prediction model, the energy consumption evaluation model dynamically monitors the process energy consumption from the perspective of hardware and code characteristics, and the energy consumption prediction model predicts the energy consumption from the static analysis.

In this model, the specific measurement is carried out through the energy consumption evaluation model, and the neural network of the energy consumption prediction model is used to fit the linear or non-linear functional relationship between code features and energy consumption. Through a large number of experimental results, it is verified that the average prediction value of linear regression model for specific types of energy consumption value model is less than 10%, and the error of prediction value for non specific types is generally higher than 20%. The average prediction value of non specific function relationship model for non-linear regression function relationship prediction model and the prediction error of energy consumption value in practical application are all within 20%. This shows the prediction validity and hypothesis difference of the two models, that is, the two models are reasonable and effective in a specific environment. This paper also describes the specific implementation of the tool.

Key Words：Green IT；Energy Consumption Analysis；Energy Consumption Prediction

原创性声明

本人郑重声明：本人所呈交的毕业设计（论文），是在指导老师的指导下独立进行研究所取得的成果。毕业设计（论文）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

作者签名：唐智强　　日期：2020 年 6 月 2 日

关于使用授权的声明

本人在指导老师指导下所完成的毕业设计（论文）及相关的资料（包括图纸、试验记录、原始数据、实物照片、图片、录音带、设计手稿等），知识产权归属大连理工大学。本人完全了解大连理工大学有关保存、使用毕业设计（论文）的规定，本人授权大连理工大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业设计（论文）。如果发表相关成果，一定征得指导教师同意，且第一署名单位为大连理工大学。本人离校后使用毕业毕业设计（论文）或与该论文直接相关的学术论文或成果时，第一署名单位仍然为大连理工大学。

论文作者签名：唐智强　　日期：2020 年 6 月 2 日

指导老师签名：　　日期：2020 年 6 月 2 日

目　　录

引　言

随着能源问题成为主流，能耗分析软件及解决方案正在变得广泛可用。计算机和其他电子设备（如智能手机、传感器）的日益增加的使用不断影响着我们的整体能源消耗。尽管信息和通信技术有助于减少其他部门（如交通或建筑）的能源足迹，但在 2020 年，其耗电量将从 2010 年的 168 千兆瓦增加到 433 千兆瓦（约 14.5%），占全球耗电量的两倍。这些数字说明了有效的电子信息及通讯技术能耗解决方案可以降低能源消耗，从而有助于减少碳排放。同时，计算机和移动设备的能源成本不断上升，也需要对计算机系统进行优化和调整。在这个领域中，绿色 IT 的研究已经提出了各种方法，旨在实现降低计算机和软件的能耗。

然而，大多数最先进的方法或只关注硬件，又或者仅提供软件的粗粒度能量估计反馈。因此，在本文中，首先采取在运行时收集细粒度的应用程序的信息，其精度与使用硬件仿真工具时的硬件监视类似，然后对收集到的信息进行模型构建，用来对其他应用程序的信息进行能耗预测。我们的方法由一个能耗评估模型（在操作系统层面上，进程与方法级）和一个深度学习预测模型组成。该方法中能耗评估模型通过操作系统从硬件设备（例如 CPU、网卡）收集的原始信息，以及从软件（CPU 时间，通过字节码插装或统计采样）获取代码运行信息并将二者综合作为代码原始能耗信息。深度学习预测模型使用前一步获取的原始能耗信息并分析其相应代码结构特征作为数据构建神经网络模型并对其他样本进行预测并验证命中率。

本文的工作扩展了先前的命题和实验，考虑到了先前的研究如编程语言和算法的影响。本文采用并更新了其中代码级别的细粒度分析方法。同时，本文还对预测和数据处理方面进行了探索，验证了几种有效的神经网络模型构建方法及优缺点，并给出了一定的高能耗优化方案。

1 绪论

1.1 课题背景与意义

在当今全球大力倡导绿色低碳环保发展经济的巨大背景下，代码设计相关能耗问题成为了一个日益地引起了人们高度关注的社会热点问题，成为了软件设计的重要考虑因素之一。软件能耗和其他因素所带来的各种环境污染问题互相连带已经逐渐成为了整个经济社会普遍讨论和关注的一个热点。从最初受到能源条件限制的嵌入式软件到如今云计算技术发展下的分布式软件，无一例外都将能耗优化作为重要技术指标。在传统的计算机软件系统中，代码的能耗是指软件 CPU、存储器（包括内存、硬盘等）以及网卡等相关器件在其运行后的一段时间内消耗的全部电能。能耗的相关方法研究存在两个重要的热点及其研究的方向，分别研究的是能耗评估与能耗优化，前者指软件在对应硬件环境或计算机系统下的能耗估计与评价方法研究，后者指降低软件能耗的相关研究。一方面，与能耗评价和优化相关的研究尚处于起步阶段，其模型、算法、策略和实施都处于不成熟阶段。因此，研究的机遇与巨大挑战并存。目前的研究主要集中在“Internet 数据中心物理能耗”或“基于资源分配和任务调度的大型分布式系统能耗优化方法”上。另一方面，以代码结构为代表的组织结构应用形式已经成为人们追求更好的算法、程序结构和低能耗的重要程序开发指标。如果能在现代程序系统设计阶段对程序系统整体能耗结构特点进行分析和综合评价，尽快准确找出系统整体能耗结构中存在的问题，调整系统结构并进行设计，则将有效地避免了在应用系统开发过程中的后期因本系统运行后自身问题导致重构，这样则可以大大提高了开发效率，降低了开发成本。这些充分证明了本文的研究成果具有良好的应用价值。最后，本研究具有对环境潜在的非常显著的生态和经济效应，降低能源消耗不仅大大节约了系统的成本和经济成本，而且大大减少了系统的碳排放，实现了环境保护，符合当前国家倡导的支持低碳发展的市场经济和社会经济政策取向。综上所述，本文的研究成果具有很强的指导性、前沿性和实用性。

本文确定了能耗研究的范围“代码模块”，并使用了“代码特征”的概念，研究了代码模块的结构与面向对象应用软件代码模块之间的相互关系，实现能耗评估与高能耗程序优化方案，从而帮助指导和开发低能耗的应用软件。面向对象的程序由类和方法组成，其中方法由经典的内部与外部逻辑结构，如循环、分支、递归、嵌套等组成，类中又有私有变量的定义与声明，其他类成员的调用等，我们称之为代码特征，代码特征会影响 CPU、内存等耗能设备以及网卡等网络通信设备的工作状态，进而影响程序的能耗。两段具有相同语义但不同代码特性的代

码会导致很大的能耗差距。主要原因如下：首先，CPU 的功率是动态的，代码的特性将直接影响 CPU 的工作状态。第二阶段是存储设备功率动态变化，代码特性将直接影响 CPU 的读写位置和数据的读取。综上所述，对于一个方法来说，程序内部的代码特性会对能耗产生影响，因此有必要对其进行评估和优化[1]。本文的目的是提出一个方法级别的能耗开源应用软件系统的评估和优化工具，总结一般概念，原则和具体方法的能耗评价和优化，从而有效地指导能源消耗评价优化应用系统的设计和应用软件的开发。

1.2 国内外研究现状

国外一些较具有国际代表性的对嵌入式软件的能耗计算和建模的研究成果摘要如下：Tan 首先提出了一种基于特征分析的宏模型能耗软件计算方法[2]。该方法利用目标处理器的能耗模型来描述能耗函数，然后根据函数的宏观模型来描述整个软件，并对软件的能耗进行评估。Tan 提出了一种系统结构转换方法[3]，从嵌入式架构的系统设计角度对于嵌入式软件的宏观能耗可靠性进行了分析和优化。该计算模型采用了线性回归计算方法，适用于各种输入模型。Lee 等人描述了一个基于过程代数的正式框架[4]，其中实时系统建模和分析的能力是有限的。Senn 等人研究了具体的 AADL 架构建模语言（即阐述了基于 AADL 的嵌入式系统能耗模型评估方法）[5]。

软件能耗建模的研究在国内起步较晚，但经过多年的艰苦研究，国内一些研究机构已经取得了一些可喜的成果。代表性著作有：张腾腾等人提出了一种基于过程代数语言的通信顺序过程(CSP)能耗模型[6]。基于该接口，模型定义了系统的最大、最小和平均能耗，但缺乏对模型的验证。陈丽琼和其他人提出能耗建模、系统模块建模、任务建模、任务间关系建模、通信协议延迟 Petri 网建模[7]，分析了系统调度性和能源消费的限制，以及启发式算法来满足可行的调度时间限制和能源消耗的约束。赵霞等人提出了一种基于微指令模型架构的嵌入式操作系统[8]。在对能耗进行定量分析的同时，他们提出了一种基于软件功能的单时钟周期能耗和能耗估算方法，分析了操作系统内核的能耗估算模型，估算了内核的执行路径和服务例程，对原子级功能的分析表明，操作系统内核系统的关键原子级能软件模块的能耗和特性对能耗有显著影响。

国内外的软件级计算机系统能耗的评估和系统优化的研究主要分为硬件与相关软件两个研究方面，其中与硬件级计算机相关的研究主要考虑计算机内核、内存、磁盘的各种能耗变化特征，借此有效地评估其系统的能耗并且合理地设计更加可靠节能的计算机硬件相关元器件，发展经过多年的积累和发展已较为的完成。国内外的软件级计算机能耗的评估和软件性能优化在相关研究和领域的应用

以及相关研究方法和工具有很多，软件级计算机能耗的研究主要分为包括计算机能耗评估和系统优化两个大部分，国内外的相关软件研究从能耗评估的层面上可以大致分为机器指令、源码算法和软件体系管理结构以及软件级三个主要研究层面。其中最核心的思路之一就是把建立在硬件能耗估算单元的操作系统级能耗通过硬件进行累加的能耗估算方式直接映射到建立软件能耗估算单元上，从建立软件能耗估算单元的角度出发入手来设计和研究建立软件能耗的操作系统优化评估问题。指令级软件能耗的估算就是通过将代码执行的每条指令能耗的平均值进行累加作为总估算能耗；语句建立指令级软件能耗的估算则将指令语句代码作为软件能耗估算单元，研究指令与语句之间的关系逻辑结构以及代码的结构特征对软件能耗的直接影响；模块级能耗估算则通过类、方法等软件模块作为估算单元，研究模块间关系对能耗的影响。这三个估算单元研究的层面分别是评估的粒度由细变粗，评估的速度由慢到快，下面分别详细介绍这三种研究层次的方法对软件操作系统能耗优化评估的方法和研究。

1.2.1 指令级能耗估算

指令级能耗估算又可分为白盒方法和黑盒方法，白盒方法与硬件环境相绑定，首先通过估计代码在目标硬件系统上生成的指令执行能耗，之后累加指令能耗估算代码能耗。可快速准确的预估代码能耗，缺点是可移植性差。黑盒方法则考虑统计全部指令执行产生的能耗以预估代码能耗，黑盒方法更像是一种“测量”而非“估算”，无法精准确定能耗与代码之间的映射关系。

1.2.2 语句级能耗估算

语句级能耗估算通过评估每个语句块（语句块划分程度由估算粒度决定）的测量能耗以及语句块之间逻辑关系（语句结构特征）对能耗的影响来静态分析代码求出能耗的估值，语句结构能够显著影响代码的能耗，语句级能耗估算也是优化代码能耗的重要途经之一。

1.2.3 模块级能耗估算

模块级能耗估算可按照方法，类和包将代码作为模块划分，模块级别的估算不仅要综合考虑与语句级相关的运行过程能耗还要详细考虑到代码和模块间的语句级关系对代码运行过程能耗的直接影响，这种关系可通过代码的具体引用方式来体现（调用、转换、多态、通信）。

本文采取的估算方法为模块级与语句级能耗估算相结合的方式。

1.3 课题研究内容

本文首先从硬件角度出发，将代码方法在硬件上产生的能耗与方法本身结合，其次从代码结构特征的角度出发以实现面向对象（Java）的软件模块的能耗评估，并给出高能耗定位与优化方案，具体研究能容如下：

（1）定义了能耗研究的代码模块粗细粒度（方法）与代码结构特征，方法为本文能耗评估的单元，代码特征为本文的研究角度。

（2）能耗模型的建立，为模块的能耗评估提供基础的理论模型，并从实验角度验证其有效性。

（3）基于能耗模型，从线性回归分析的角度设计神经网络模型对代码能耗进行预测，从理论和实现角度验证评估方法的正确性、有效性。

（4）基于能耗模型，从非线性离散回归的角度利用 BP 神经网络模型对代码能耗进行预测，并验证其命中概率。

（5）设计能耗评估工具，作为开发者代码能耗评估插件使用，对能耗进行预测并给出优化方案，满足能耗需求。

1.4 论文组织结构

本文整体分为六章，各章节简介如下所示：

第一章"绪论"，对与本文所具体阐述的主要内容的相关基础研究课题学术内容特点进行了具体说明，主要研究内容具体包括了本课题的主要基础研究课题背景、国内外的基础研究课题发展趋势现状、课题主要基础研究的学术内容等。

第二章“进程级能耗分析模型”，提出能耗评估的基本理论和模型，即适用于软件模块级的能耗测量模型。为之后估计模块的能耗以及分析、评价其能耗情况提供了理论依据。

第三章“基于神经网络的能耗预测模型”，首先定义了代码结构特征，提出了以代码结构特征模型作为能耗预测模型的理论分析基础，其次提出了基于神经网络理论中的基于深度学习机器智能学习的预测模型。

第四章“能耗工具实现”，该章设计与实现了面向 Java 语言的能耗评估与预测工具。该工具包括了能耗评估、能耗分析初始化、能耗预测、能耗评价与优化方案四个部分。

第五章“有效性验证”，对前几节模型进行从理论性到实验数据的有效性验证

第六章“结束”，该章首先对全文的工作进行了概括，接下来指出了今后进一步的研究方向。

2 进程级能耗分析模型

本章提出了软件模块的“能耗分析模型”，包括了能耗评估模型和能耗映射模型两部分。能耗评估指通过软件手段对软件模块的能耗值进行测量计算，属于“白盒分析”。能耗映射是在能耗评估的基础上，从运行时收集到的软件层面信息，将能耗定位到语句，从而指导模型设计[9]。

2.1 模型原理

一般情况下软件的总能耗是由如下公式计算得到的：

$$E_{software} = E_{comp} + E_{com} + E_{\inf ra} \tag{2.1}$$

其中，$E_{software}$ 为软件总能耗，E_{comp} 为计算成本（即，CPU 处理，内存访问，I/O 操作），E_{com} 是通过网络交换数据的成本，$E_{\inf ra}$ 是操作系统和运行时平台（如 JVM）产生的额外成本。本文的模型基于类似的原理，考虑了功耗计算的模块化方面（例如，不同硬件组件的功耗总和）。基础设施电力（$E_{\inf ra}$）包括在电力模型和原型的计算成本中。功率（瓦特）是计算单位时间（1 秒）的能量消耗（焦耳），功率（瓦特）计算为每单位时间（1 秒）的能量消耗（焦耳）。由此，可将计算公式概括为：

$$P_{software} = P_{comp} + P_{com} \tag{2.2}$$

在此阶段，本文定义了两个模型，一个用于 CPU 计算成本，另一个用于网络通信成本。其中 P_{comp} 等于软件消耗的 CPU 功率，P_{com} 等于网卡传输软件数据所消耗的功率。接下来，将详细介绍在能耗评估模型中使用的 CPU 和网络电源模型。

2.2 能耗评估模型

2.2.1 CPU 能耗评估模型

特定应用程序消耗的 CPU 功率（以进程 PID 举例）可以用以下公式表示：

$$P_{CPU}^{PID}(d) = P_{CPU}(d) * U_{CPU}^{PID}(d) \tag{2.3}$$

$P_{CPU}^{PID}(d)$ 是在给定的时间内，特定进程 PID 消耗的 CPU 功率多少，$P_{CPU}(d)$ 是整个 CPU 的功耗 $U_{CPU}^{PID}(d)$ 是在此期间整个 CPU 的使用情况。因此，我们的方法是估计 CPU 执行进程 PID 所需的功率。这是通过计算 PID 在给定持续时间 d 内的总体 CPU 功率以及 CPU 使用率百分比来实现的。

2.2.2 CPU 总功率

大多数现代处理器遵循如下标准公式：

$$P_{CPU} = \mathrm{c} * f * V^2 \tag{2.4}$$

其中 f 为频率，V 为电压，c 为取决于硬件材料（如电容和活度因子）的常数。由于动态电压和频率缩放（DVFS）以及功率依赖于处理器的电压（以及随后的频率）等原因，功耗并不总是线性地依赖于 CPU 利用率的百分比。因此，一个简单的 CPU 利用率分析器不足以监视功耗。此能耗分析模型考虑了 CPU 的这些方面，并允许精确地监视功耗。

根据公式（2.4），计算给定时间内总的 CPU 功率等于计算一个静态部分（常数 c）和一个动态部分（频率 f 及其相关 V）。对于静态部分，常量是一组描述物理 CPU 的数据（例如，电容、活动因子）。制造商有可能会提供这个常量，但在大多数情况下，这个值是缺失的。为了解决这个问题，决定使用处理器的总功率与其热设计功率（TDP）值之间的现有关系。TDP 表示计算机冷却系统消耗处理器产生的热量所需要的功率。因此，总的 CPU 功率可以与 TDP 相关联，如下式所述：

$$P_{CPU}(\mathrm{f}TDP, VTDP) \approx 0.7 * TDP \tag{2.5}$$

其中 $fTDP$ 和 $VTDP$ 分别表示 TDP 状态下的处理器频率和电压。使用这种方式的好处是大多数厂家可提供相关技术参数。在我们的架构中，TDP 存储在能耗模型的本地数据库中。对于动态部分，频率 f 与一个特定的电压 V 相关联，一个给定的电压与一个或多个频率相关联，因此当电压降低时，频率就会改变。反之改变频率电压也会随之改变，尽管在某些情况下，单个电压可以支持多个频率。处理器使用的频率由操作系统 API 提供，而电压由制造商提供。

2.2.3 进程的 CPU 使用率

为了计算给定进程（由其 PID 标识）的 CPU 使用率，本文采用计算此 PID 的 CPU 时间与持续时间 d 内的全局 CPU 时间（处理器对所有进程处于活动状态

的时间）之间的比率，所以一个进程的 CPU 功耗等于每个频率的 CPU 功耗与所有频率的 CPU 时间的平均值：

$$P_{comp} = \frac{\sum_{f \in frequencies} P_{comp}^{f} * t_{CPU}^{f}}{\sum_{f \in frequencies} t_{CPU}^{f}} \quad (2.6)$$

2.2.4 网络电源模型

进程的网络功率使用类似于 CPU 功率的公式计算。模型基于运行时收集的或者由制造商的文档提供的信息。作为第一步，首先可以使用线性方程的类似模型可以应用于无线网卡[10]。我们从制造商的文档中得到，根据给定的网卡吞吐量模式（例如 10 MB），可以得知在一定时间内（通常为 1 秒）传输字节所消耗的功率（瓦特）。因此，网络功率模型定义为：

$$Power_{process}^{Network} = \frac{\sum_{i \in states} t_i * P_i * d}{t_{total}} \quad (2.7)$$

其中 $Power_{process}^{Network}$ 为状态 i（厂家提供）网卡消耗的功率，d 为监控周期的持续时间，为使用网卡传输数据所花费的总时间。

2.3 能耗映射模型

本节包含“能耗映射模型”相关内容，介绍了从硬件层面上将能耗定位到对应代码区域的（能耗映射）使用的 CPU 和网络电源模型，以及从操作系统层面上使用相关监测技术（字节码插装，统计抽样），为能耗评估模型实现的关键核心。

2.3.1 CPU 映射模型

利用从分析应用程序和监视系统收集的信息，可以计算出每种方法的合理的 CPU 使用时间。本文使用这个信息来计算每个方法和线程消耗的 CPU 功率。因为应用程序代码通常是在线程内部执行的。所以首先计算每个线程的功耗。为此，应用以下公式：

$$Power_{thread}^{CPU} = \frac{Time_{thread}^{CPU} * Power_{process}^{CPU}}{Duration_{cycle}} \quad (2.8)$$

其中 $Time_{thread}^{CPU}$ 是上一个监视周期中线程的 CPU 时间（从平台环境中获得，比如 OS 或 JVM），$Power_{process}^{CPU}$ 是上一个监视周期中应用程序进程消耗的电量（从

能耗评估模型获得），$Duration_{cycle}$ 是监视周期的持续时间。然后我们过滤这些方法，以获得在最后一个监视周期中运行的方法的列表（不管它们是否仍在运行）。对于每个线程，我们获取它从列表中调用的方法（线程通常有自己的执行堆栈，由帧组成），一个框架表示一个方法调用。此外，我们使用以下公式对每种方法的 CPU 时间进行了比较准确的估计：

$$Time_{method}^{CPU} = \frac{Duration_{method} * Time_{thread}^{CPU}}{\sum_{m \in Methods} Duration_m} \tag{2.9}$$

其中，$Duration_{method}$ 为方法在最后一个监控周期的执行时间，所有方法在最后一个监控周期后的执行时间之和为 $\sum_{m \in Methods} Duration_m$。最后，我们用这个公式计算每一种方法的功耗：

$$Power_{thread}^{CPU} = \frac{Power_{method}^{CPU} * Time_{thread}^{CPU}}{Duration_{cycle}} \tag{2.10}$$

2.3.2 网络电源映射模型

本文使用应用程序传输的字节数来计算网络功率。我们首先计算在最后一个监视周期中读取/写入的字节数。然后，我们从能耗评估模型收集应用程序进程消耗的网络电能。因此，每一种方法消耗的网络功率为：

$$Power_{method}^{Network} = \frac{Bytes_{method} * Power_{process}^{Network}}{Bytes_{process}} \tag{2.11}$$

其中 $Bytes_{method}$ 是该方法读写的字节数，$Power_{process}^{Network}$ 是应用程序消耗的能量，$Bytes_{process}$ 是该应用程序的所有方法读写的字节数。因此，每个线程的网络功耗是线程中运行的所有方法的网络功耗的总和，如下式所示：

$$Power_{thread}^{Network} = \sum Power_{method}^{Network} \tag{2.12}$$

2.3.3 操作系统进程监测模型

操作系统进程监测模型是一个软件级的概要分析架构，它负责分析运行的应用程序和评估它们的性能。可以使用数种行之有效的分析技术，本文采用的是字节码插装或对应用程序进行采样，这些方法各有优缺点。

统计抽样：统计抽样并不修改应用程序的代码，而是提供应用程序的能耗概述。但是，它不如字节码插装工具精确。

字节码插装：字节码插装将分析代码注入到应用程序的代码（或字节代码）中，因此允许分析器捕获与能源消耗相关的所有必要事件。另一方面，检测添加额外代码，增加了运行额外代码的开销。这个开销取决于添加的代码，可能是常量，也可能在执行过程中发生变化（无论添加的代码是否执行与方法相关的任务，例如用数据填充到图中）。

本文的映射方法没有指定一种分析方法。实际上倾向于在实现过程中将其作为一种技术选择，并使用专用的 API（能耗模型以及操作系统提供的）与分析器和低级监视环境进行通信。

2.3.4 模型实现流程

（1）操作系统进程监测模型在运行时监测应用程序，收集有关其资源利用率的统计信息并分类为更细的粒度，方法持续时间，CPU 时间，或者通过网卡传输的字节数等。

（2）进行关联过程，以将特定于应用程序的统计信息与过程级电源信息关联起来。

（3）将每个方法或每个线程的功耗信息将显示给用户，并可以作为服务公开。

3 基于神经网络的能耗预测模型

Java 是一种面向对象的分布式编程软件设计语言，虽然硬件的不断升级与新标准 Java 性能的调整在一定程度上改善了性能问题，但仍然存在相当大的问题。除性能指标外，能耗指标正在成为另一个程序设计的重要指标，本文这部分将首先分析面向对象软件的能耗特征，分别建立不同的能耗预测神经网络模型，然后提出具体的能耗优化方案，这一部分将从两个方面：特征和神经网络模型来解决上述问题。

3.1 代码能耗特征分析

本小节从影响代码能耗的特征结构的角度出发，提取了有效代码行数、构件个数、算法复杂度、调用深度、耦合程度这 5 个代码体系结构级特征度量[11]。本小节具体分析了上述的代码结构特征对代码能耗的影响过程，并且详细说明如何对上述软件特征模型进行有效的度量。

3.1.1 有效代码行数

Java 程序代码在编译过程中首先被翻译成字节码文件，然后由 JVM 编译成对应硬件平台的指令。硬件平台根据二进制指令完成相应操作。在执行过程中，取指令和取操作数阶段涉及到内存读写访问和总线数据传输，译码阶段涉及到微处理器内部算术运算和逻辑运算，执行阶段根据不同指令类型完成不同的操作，如内存读写、I/O 读写等，而整个过程中，微处理器内部某些寄存器的值也发生对应的改变。以上一系列操作的所有步骤都会触发相关电路活动，并产生相应的电路能耗。可见，程序代码作为机器指令的来源，是产生软件能耗的根本原因。实验表明，每个二进制指令在一个固定的平台上具有一个固定的能量消耗值。执行的指令越多，相应的能量消耗就越大。Java 软件代码在固定编译工具和硬件环境中生成的二进制代码是固定的。因此，可以认为软件代码的行数决定了软件能耗的大小。软件代码分为有效代码和无效代码。有效代码是指可以被编译器转换成可执行机器指令的代码，否则就是无效代码。通过测量各种代码操作在运行过程中的能量消耗变化，得到以下四种类型的操作作为有效代码：变量计算、变量赋值、函数初始化和函数调用。为了提高有效代码行的测量精度，测量过程中的具体代码将被转换，步骤为：

（1）如方法中出现选择分支结构，设置相应选择概率乘以有效代码行。

（2）对于组件中的一个循环，将循环展开到相应的变量赋值和变量操作语句，然后计算有效代码行数。

（3）在类继承的情况下，在计算子类的有效代码行数时，将父类的公共部分加入到有效代码行的统计中。

（4）以 friend 为例，在计算 friend 类的有效代码行的时候，在有效代码行的统计中加入基类的保护部分和公共部分。

（5）对于虚函数，只计算在子类中有效的代码函数。根据功率测试。

同时，变量计算、变量赋值、函数初始化和函数调用这四种有效代码产生的指令能耗在不同平台各不相同，在计算有效代码行数时应计算分别的权重再累加。

3.1.2 构件数量

构件是指类内部或方法内部定义的变量，传参或对象实例，以及静态变量或常量等定义。可以认为，构件数量决定了软件所包含的模块数量。这些都是会通过占用内存产生能耗的主要部分，一个由 Java 编译的程序占用的内存主要分为以下几个部分：

（1） 栈区（stack）：由编译器自动分配释放，存放函数的参数值，局部变量的值等。

（2） 堆区（heap）：由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收

（3） 全局区（静态区）（static）：全局变量以及静态变量位于同一储存区，初始化完成的全局变量和静态变量与未初始化的全局变量和未初始化的静态变量位于相邻的两处区域。在程序结束后由都将被系统自动释放。

（4） 文字常量区 ：储存常量字符串等静态常量，在程序结束后将被系统自动释放。

（5） 程序代码区 ：存放函数体转换生成的二进制代码。

综上所述，构建的数理计算由于其在内存中的活动期和位置各不相同，在计算构件个数时应分别计算或根据硬件平台的不同设置权重求和。

3.1.3 圈复杂度

圈复杂度是代码复杂度和方法复杂度的度量。它可以用来度量模块内部结构的复杂性，可定量地表示为独立的当前路径的数量，或者理解为覆盖所有可能场景的最少使用的测试用例的数量。较高的圈复杂度表明程序代码的判断逻辑是复杂的、低质量的并且难以测试和维护。它还间接影响代码的能量消耗（大量逻辑相关指令）。程序中潜在的错误与高圈复杂度也有很大关系。圈复杂度主要与分支语句（if、else、switch 等）的数量正相关。当一段代码包含更多分支语句时，

逻辑复杂性会增加。在圈复杂度的计算中，可以得到程序控制的流程图。通常使用的计算公式是：

$$V(G)=e-n+2 \tag{3.1}$$

e 代表在控制流图中的边的数量（对应代码中顺序结构的部分），n 代表在控制流图中的节点数量，包括起点和终点（1、所有终点只计算一次，即便有多个 return 或者 throw；2、节点对应代码中的分支语句）。

3.1.4 调用深度

调用深度是指软件在执行路径中方法的位置深度，是方法的外部复杂度度量。程序顺序执行的过程可以视为程序执行路径，执行过程中参与运行的方法可以视为路径节点，不同的执行过程可以视为软件不同的分支路径。路径长度越长，表示参与运行的方法数量越多，调用深度则越高，则父方法函数活动期的会间接增加代码的能耗。可以认为调用深度对软件能耗有间接的影响。不同的方法和分支调用路径，路径的长度不同，对应的活动期软件的能耗也各不相同，由此可见，调用深度能够通过影响实际处于活动期的方法个数对软件能耗产生间接影响。调用深度的度量根据系统执行的具体流程可以分为两种情况：单一路径调用深度度量和多分支路径调用深度度量，以下将分别介绍：

（1）单一路径：方法的调用深度等于路径执行过程中各方法所在路径节点。

（2） 多分支路径：多分支路径的调用深度度量情况相对复杂，可能会出现以下两种情况。一、多分支路径出现回路情况，这种情况下将回路分解并分解为多条路径，保证路径各个节点都只出现一次，并且由于回路的出现会降低软件可靠性，在优化方案中应给予说明。二、多分支路径方法重复调用的情况，这种情况下将对应方法分为两个副本分别考虑其对父方法的间接能耗影响。

3.1.5 平均耦合程度

耦合度的概念最早被广泛使用在了软件系统工程的技术领域，主主要是用于描述和衡量一个软件模块与另一个软件模块的耦合和逻辑紧密的程度耦合度的概念表达了软件结构中各个模块之间相互关联的程度，模块间耦合度越高，系统越复杂，对于模块的理解、维护和设计以及调试的工作难度也越高。合理的软件系统结构应该是追求系统的低耦合，高内聚。耦合度的体系结构由软件工程衍生出来，耦合度的概念仍然可以作为代码重要的质量属性，衡量方法间的复杂程度。方法间耦合程度的高低决定了方法接口的复杂度的大小，方法间耦合程度越高，则方法被执行的有效运行代码越多。可见，构件间耦合程度能够通过方法的被执行次数提高来间接影响软件的功耗。

判断两个构件模块之间是否可能存在上述耦合依赖关系，即是通过判断两个构件是否可能存在耦合控制交互或者是否存在数据交互，如果两个构件存在控制交互或者没有数据交互，则两个构件模块之间存在上述耦合依赖关系；反之，则不可能存在上述耦合依赖关系。构件模块间的耦合度依赖关系是指两个构件模块之间的耦合性依赖关系，包括控制关系、调用依赖关系、数据的传递依赖关系。也或者可以是通过直接判断两个构件模块之间是否可能存在上述耦合依赖关系来直接确定它们之间到底是否可能存在耦合关系。

3.2 神经网络模型

3.2.1 线性预测模型

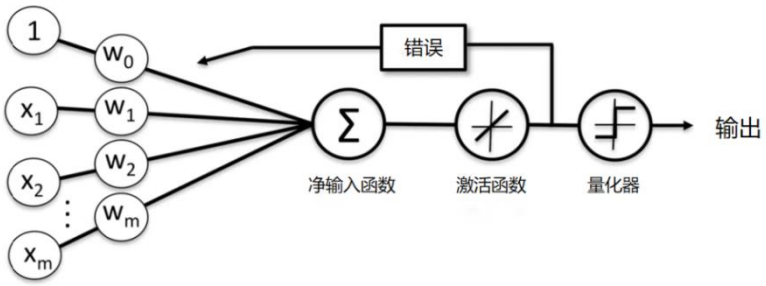

图 3.1 自适应线性网络结构图

代码结构特征量与代码能耗存在明显的正相关且连续倾向，所以可假设代码结构特征与代码能耗存在线性关系，为了更加准确地描述这种线性关系，选择一种合理且有效的数值逼近模型是十分必要的。本小节采用使用较为广泛的自适应线性网络对代码结构特征与软件能耗进行拟合，网络流程图如图 3.1 所示。

本模型采用的激活函数为线性函数，每次输入的训练集，可以作为调节误差的手段，其功能是将期望输出与实际输出相比较，得到一个训练集的误差信号，以此来调节权值，以保证最近保持期望输出与实际输出相等。该模型采用 LMS 即最小二乘法作为学习规则，采用代价函数为均方误差函数，计算公式为：

$$J(w)=\frac{\sum_{i} y^{(i)}-\phi(z^{(i)})^{2}}{2} \tag{3.2}$$

使用该代价函数的好处是可以微分，为凸函数。并且可以用梯度下降的方法找到误差最小的权值（每次算法循环相当于下降一步，下降一步的步幅取决于学

习率，与图中的权值点的切线斜率相关，每次权值逼近均方误差最小点的过程就是梯度下降，如图 3.2 所示）。

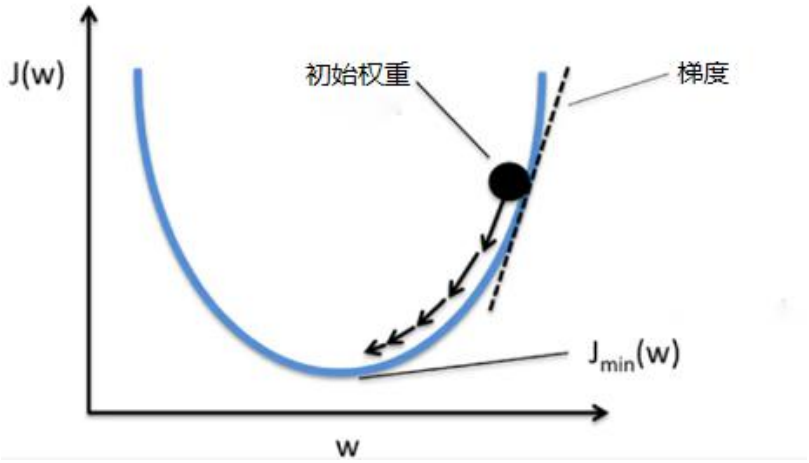

图 3.2　梯度下降算法演示图

最终的权值更新公式如下：

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \mu \sum_i (y^{(i)} - \phi(z^{(i)}))x_j^{(i)} \tag{3.3}$$

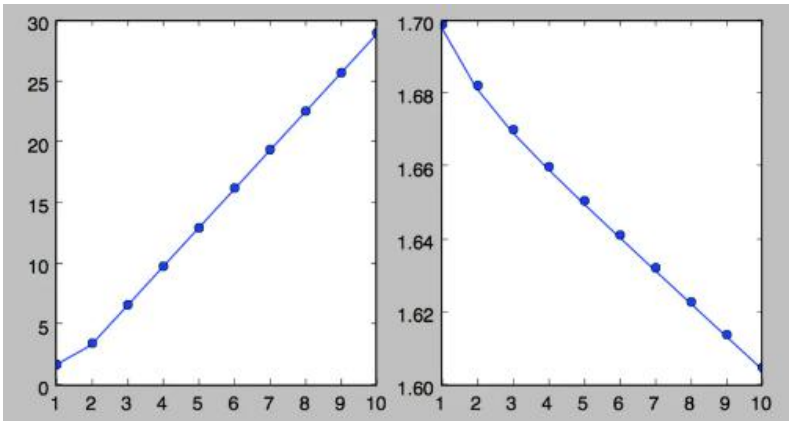

图 3.3　不同学习率对比图

学习率的影响和选择：学习率设置为 0.01 的时候，结果如图 3.3 左，均方误差最小的点是第一个点，然后越来越大。当学习率设置为 0.0001 的时候，结果如图 3.3 右，误差在逐渐减小，但是没有收敛的趋势。偏大偏小都会大幅降低算法效率。所以采取的方法是进行数据标准化。经过标准化的数据，会体现出一些数学分布的特点。标准化后，就可再次使用 0.01 的学习率进行训练分类。

3.2.2 非线性预测模型

影响代码能耗的成因非常复杂，与硬件平台的多种能耗影响因素都有关。所以我们可合理地假设代码的结构特征与神经网络代码软件能耗的关系存在非线性的关系[12]，为了更加精确地分析和描述这种非线性的关系，选择一种合理且有效的数值逼近模型是十分必要的。BP 神经网络作为一种数值逼近方法，在数学上能够逼近任意非线性函数，具有较高的拟合度，因此，本小节采用 BP 神经网络对代码结构特征与代码能耗进行拟合。模型结构如图 3.4 所示：

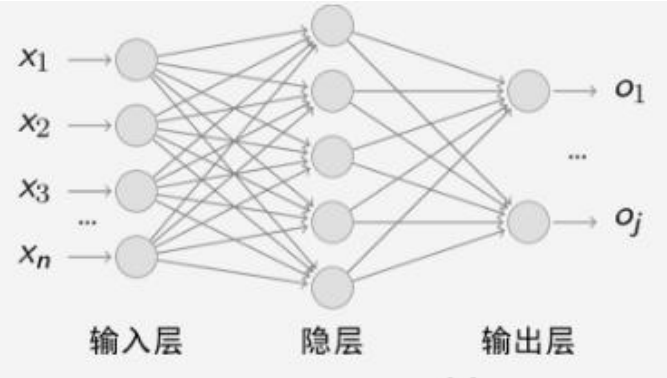

图 3.4 能耗评估模型模块图

在得到样本特征向量后输入样本，然后根据权重向量获取传感器的输入值，利用 sigmoid 函数（也称为 Logistic 函数，该函数可将任意数压缩至（0，1）区间内）来计算每个传感器的输出，然后输出作为下一层感知器的输入，依此类推，直到输出层。对于每个训练样本，执行以下操作：

（1） 首先，从前向后计算实例的输入，得到输出层中每个单元的输出。然后从输出层向前计算每一层中每个单元的误差项。

（2） 接下来，计算输出层中每个单元 k 的误差项。然后，对于网络中的每个隐藏单元 h，计算错误项并最终更新每个权重（符号描述：x_{ji} 是从节点 i 到节点 j 的输入，w_{ji} 是相应的权重值，输出表示在输出层中设置的节点）。

整个算法类似于 delta 规则的随机梯度下降算法。算法分析如下：权值的更新与 delta 规则相似，主要取决于学习率和权值对应的输入项和单位误差项。对于输出层单元，误差项是（t-o）乘以 sigmoid 函数的导数 o_k（1-o_k），这与 delta 规则误差项（t-o）不同。对于隐藏层元素，由于没有直接的目标值来计算隐藏层元素的误差，因此需要间接计算隐藏层的误差项。每个单元的隐藏层单元 h 影响的误差通过加权求和，每个误差的权重为 w_{kh}，即隐藏层单元 h 对输出的权重 k。

4 能耗分析工具实现

能耗模型为提出对软件代码能耗的评价提供了理论基础，预测模型为软件代码生成能耗的预测提供了理论依据。在此基础上，本文提出了一种软件能耗评估或预测的方法。本章设计了 Java 代码能量能耗分析工具 Java-profiler，并进行了相应的原型系统设计，实现了所需的功能。该工具包括能耗评价、能耗分析初始化、能耗预测、能耗评价和优化方案四个部分。

4.1 Java-profiler 总体设计

能耗分析工具包含了四个部分，本节将会对工具的总体设计进行简单介绍，之后几节会针对具体的每个部分进行详细的介绍。

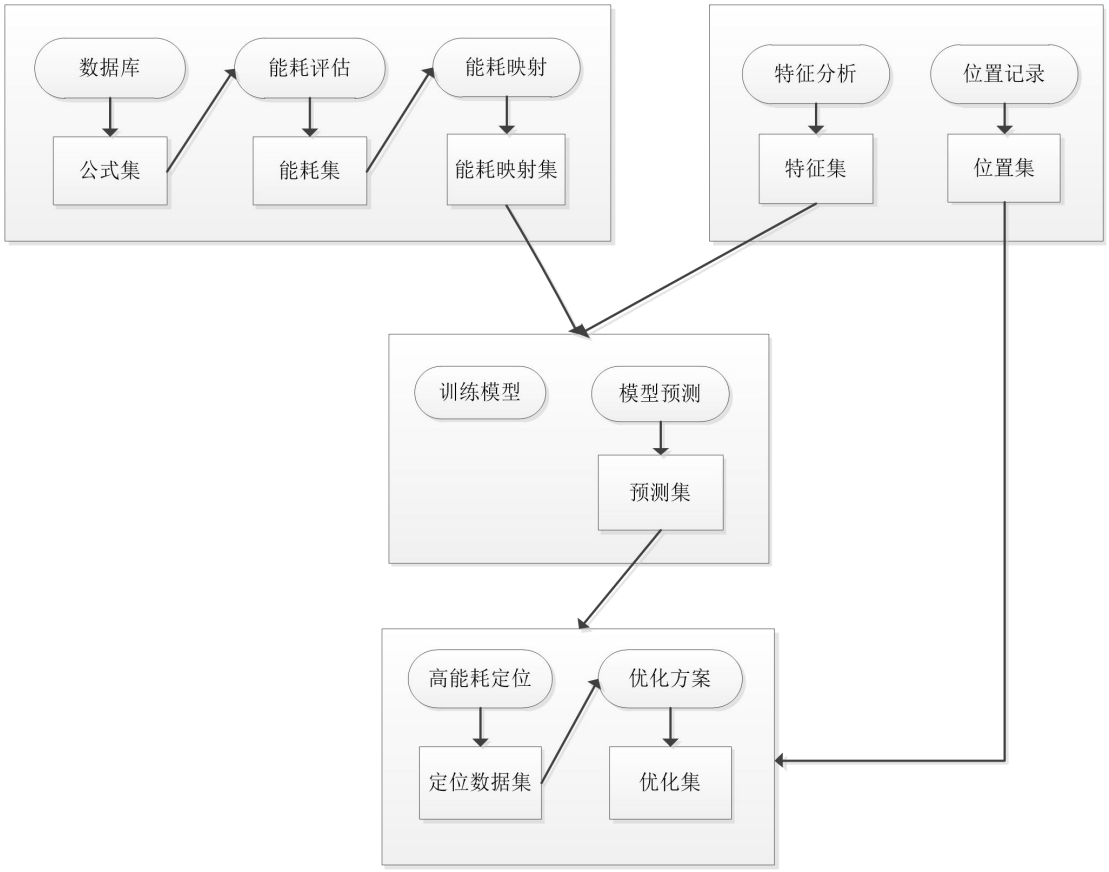

图 4.1 Java-profiler 模块功能图

如图 4.1 所示，每个系统的圆角矩形代表的是各部分中的子模块，矩形代表的是数据或者图表等结果集，由相连接的子模块生成。部分与部分之间、子模块与子模块之间都存在着依赖关系，下面对四个部分进行简要描述：

4.1.1 能耗评估

该部分的主要功能是对运行的程序进行能耗分析与能耗定位，既是分析工具的核心部分与能耗分析初始化部分共同构成能耗预测的基础部分，包含了能耗模块与能耗映射模块以及数据库模块（非核心功能，下文略）。能耗模块对运行中的程序从运行开始进行动态监测直至运行结束，收集相关进程中方法的能耗信息，并利用数据库模块提供的能耗公式计算转换为能耗评估值。能耗映射模块同样对运行中的程序从运行开始进行动态监测直至运行结束，获取程序相关方法的使用时间信息，结合能耗模块获取能耗评估模块收集转换的能耗数据集，生成运行程序的总能耗数据集及内部方法的能耗数据集。数据库模块主要储存能耗计算公式以及工具运行过程中需要保存的数据。

4.1.2 能耗分析初始化

该部分的主要功能是程序代码中方法的结构特征分析以及方法的位置记录，是能耗分析工具的另一个基础部分，包含了特征分析模块与信息记录模块。特征分析模块对目标程序按方法分别计算（有效代码行数、构件个数、算法复杂度、调用深度、耦合程度）特征值，生成特征集。信息记录模块主要收集方法的所在位置等信息，以便于之后优化使用。

4.1.3 能耗预测

该部分是分析工具的核心部分，主要功能是训练模型或者根据已有模型对代码的能耗进行预测，包含了模型训练模块与模型预测模块。模型训练模块获取能耗评估部分以及初始化部分生成的训练集或本地文件记录对相应的模型进行训练，模型预测模块对目标代码通过初始化部分生成的预测集进行预测并给出相应预测值。

4.1.4 能耗评价与优化方案

该部分是工具的扩展部分，主要功能是审查模型预测得出的预测值，找出高能耗区域并给出优化方案，包含了高能耗定位模块与优化方案模块。高能耗定位模块评估能耗预测模块的预测值并找到产生高能耗的方法位置，优化方案模块对找到的高能耗方法根据其代码结构特征给出优化方案[14]。

上述的四个部分互相依赖，互相关联，可以实现完整的软件级代码能耗分析，具体的流程图如图 4.2 所示：

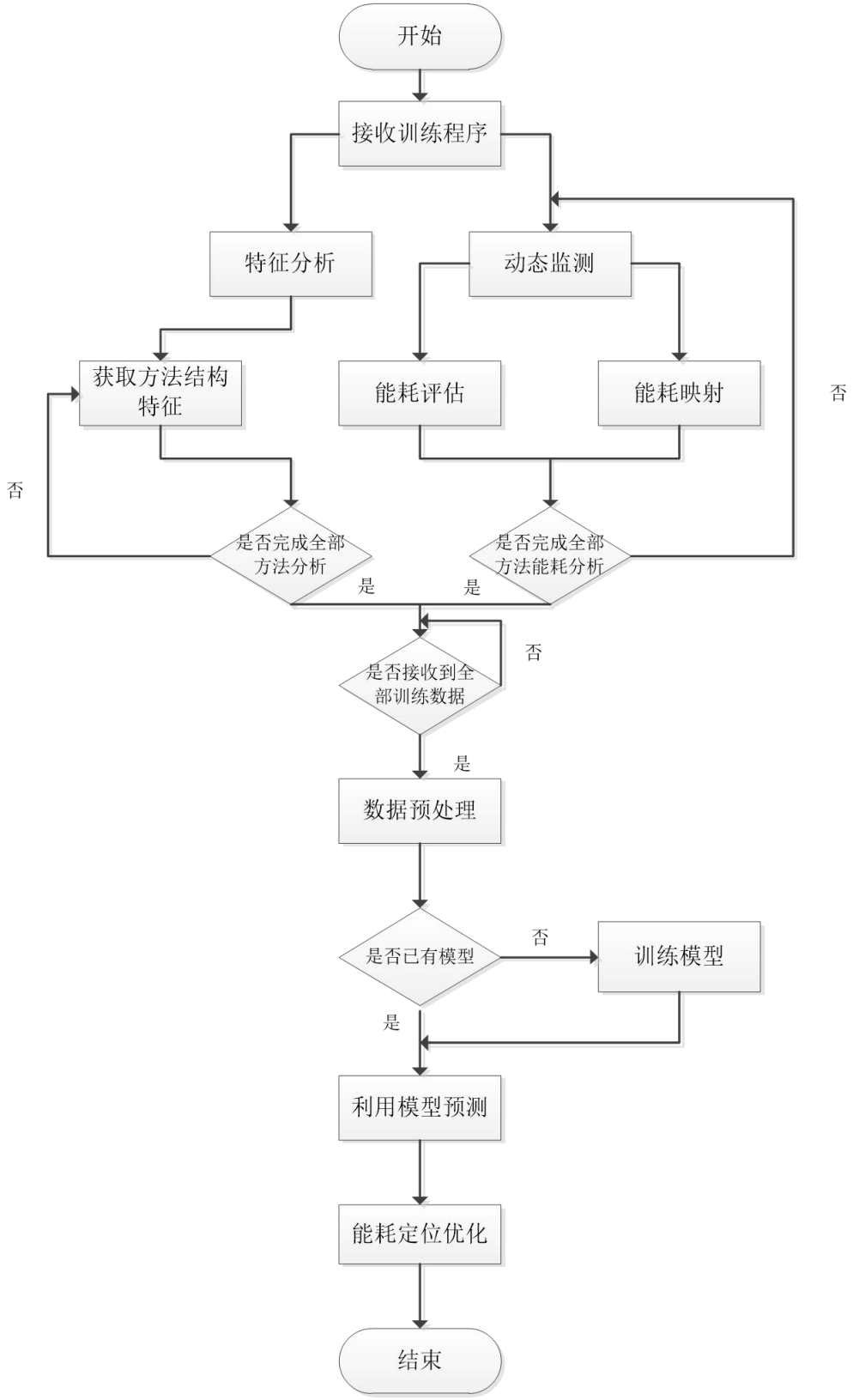

图 4.2　Java-profiler 流程图

首先基于分析环境，获取能耗评估部分所需硬件参数，以此为基础进行能耗的动态监测，得出目标训练程序的能耗监测值，并通过映射手段将产生的能耗与对应的方法相联系。接着进行模型训练，通过动态监测对目标训练程序集获得的能耗训练集，采用对应模型训练方法进行训练。之后利用训练好的模型对目标预测程序进行能耗预测，如果使用之前训练完成的模型则可不进行能耗评估步骤直接进行预测。最后，分析预测值定位高能耗代码位置并根据方法结构特征给出优化方案。下面将对耗监测、能耗分析初始化、能耗预测、能耗评价与优化方案四个部分的实现进行详细描述

4.2 能耗监测实现

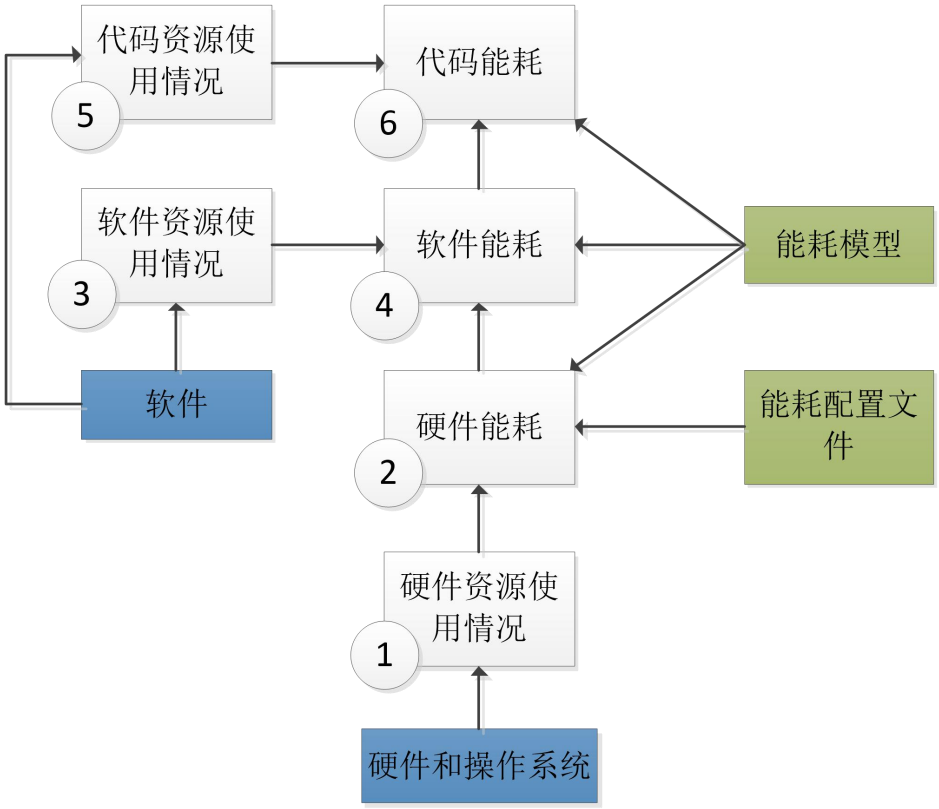

图 4.3 能耗模型流程图

能耗模型由两个不同但互补的部分组成：一个为能耗评估模型；一个为能耗映射模型。这两个部分相互协作，以便在应用程序级别（线程和方法级别）提供准确的运行时能量信息。图 4.3 描述了该方法的流程图，图中用数字标记了方法的每个步骤的顺序。蓝色框表示应用程序和硬件等现有环境，而绿色框表示我们用来估计能耗的数据和模型。其余的框表示架构的每个步骤。这些步骤在之前提

到能耗模型和映射模块中实现。具体来说，我们首先从监视硬件资源利用率开始（图 4.3 中的步骤 1），例如，我们监视 CPU 的实际频率和电压。在步骤 2 中，我们使用步骤 1 中收集的信息以及关于硬件的构造器文档，以便使用我们的能量模型估计硬件的能量消耗。在步骤 3 中，我们应用与步骤 1 相似的监视策略，以便监视应用程序的资源利用率。然后，步骤 4 使用我们的能量模型估计软件的能量消耗，以及步骤 2 计算的能量消耗。最后，在第 5 步监控源代码级别的资源利用率，并在第 6 步使用能耗模型和在第 4 步计算的能源消耗来估计它们的能源消耗。

4.2.1 能耗评估模型实现

能耗评估模型可以在运行时以系统进程的粒度监视软件的功耗，因此，与使用硬件功率表相比，可以对每个进程的功耗进行监视，并对其进行很好的评估。能耗评估模型实现图 4.3 中的步骤 1 到 4。该模型还提供了基于硬件资源的能量差异值，例如给出进程在 CPU、网络或其他受支持的硬件资源上消耗的能量。应用程序的总功耗为执行该程序进程的能耗总和，如果一个进程正在使用系统调用，那么只有当操作系统在同一个被监视的进程中执行这些调用时，这些调用所消耗的电能才会包含在进程的电能中。如果这些调用是在单独的进程中执行的，那么它们就不包括在主监控进程的结果中。特别是，如果应用程序的一个方法的调用系统调用操作系统的可用的 API 的一部分，然后由系统调用计算能量消耗作为应用程序消耗的，当且仅当操作系统执行这个系统调用相同的流程应用程序。

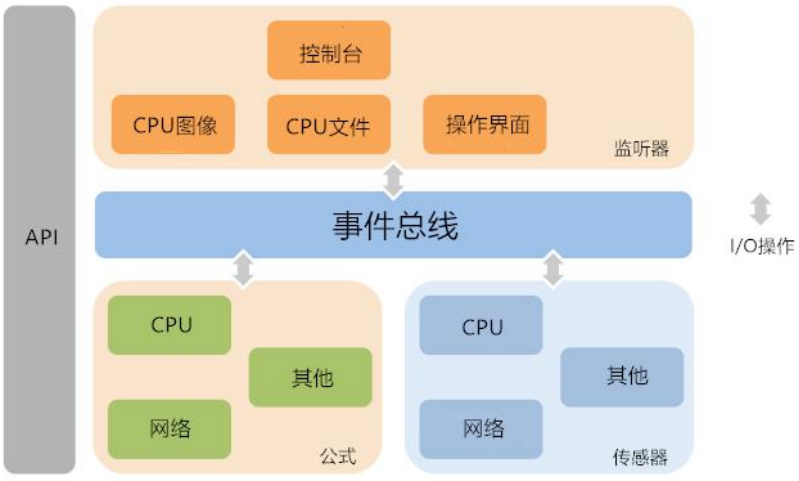

图 4.4 能耗评估模型模块图

能耗评估模型的架构是模块化的，因为它的每个组件都表示为一个电源模块（如图 4.4）。它使用带有发布/订阅范例的事件总线，以便模块进行通信。

（1）传感器模块负责收集操作系统和硬件相关信息。例如，CPU 传感器收集 CPU 在每个处理器频率（支持 DVFS 时）上所花费的时间，用于监视进程。

（2）将此信息发布到事件总线中的所有侦听模块。第二步是数据处理模块监听已发布的传感器数据，并估计所监视过程的功耗。公式模块在计算中使用数据库查询获得的硬件特性（如每个频率的 CPU 电压，通常由硬件构造器提供）。

（3）监听器模块监听事件总线并收集公式模块发布的数据。因此，CPU 文件监测器模块负责将所监视进程的 CPU 功耗写入文件。该模型允许用户查询不同的模块，获取传感器、公式等信息，或者直接通过监听器获取能耗。

能耗模型是在 scala4 中实现的，它是基于事件驱动的体系结构，同时运用了基于使用 Akka 库的模块化异步事件驱动体系结构。架构集中在一个公共事件总线上，每个模块都可以发布/订阅发送事件。这种体系结构的一个特殊之处在于，每个模块都处于被动状态，并对公共事件总线发送的事件做出响应。

最终能耗模型可提供了一个模块化的库，其中只有部分组件是平台相关的。它的模块化允许轻松地移植库，同时保留大部分建模代码。应用程序可要求能耗模型通过提供进程 ID （PID）来开始监视进程，然后能耗模型将加载所有模块，并开始监视进程 PID，同时将能耗值写入一个文件，每个区间值为 500 ms。

4.2.2 能耗映射模型实现

能耗映射模型负责分析正在运行的应用程序，并估计它们在更细粒度（即线程或方法级别）上的能耗。模型执行图 4.3 中的步骤 5 和 6。分析部分在运行时分析应用程序，收集有关其资源利用率的统计信息，例如对于应用的每个方法，收集方法持续时间、CPU 时间或通过网卡传输的字节数，以更精细的粒度被收集和分类，接下来，将发生关联阶段，以将特定于应用程序的统计信息与过程级电源信息关联起来。三。最后，每方法或每线程功耗信息显示给用户。

能耗映射模型使用能耗评估模型提供的每个进程的电源信息，并将其与从应用程序监控中收集的信息相关联，以提供每个方法的电源信息。映射模块实现为一个 Java 代理，它在启动时挂接到 Java 虚拟机，并开始监视和收集被监视应用程序的能源相关信息。能耗映射模型具有两个版本，每个版本在收集和关联信息方面采用不同的方法。第一个版本使用字节码插装，而第二个版本使用统计抽样。在监测软件的能耗时，每一种方法都有其优点和缺点，并且在某些情景中可以比其他方法更好地应用。接下来，我们将详细解释每个实现。

（1）字节码插装实现

能耗映射模型的插装版使用字节码插装技术收集资源使用信息。本文中使用ASM (http:// asm.ow2.org/;）将监控代码注入到遗留应用程序的方法中。asm6 是一个使用 Java 字节监控器代码的操作和数据分析的框架。仪器的过程如下，如图 4.5 所示。

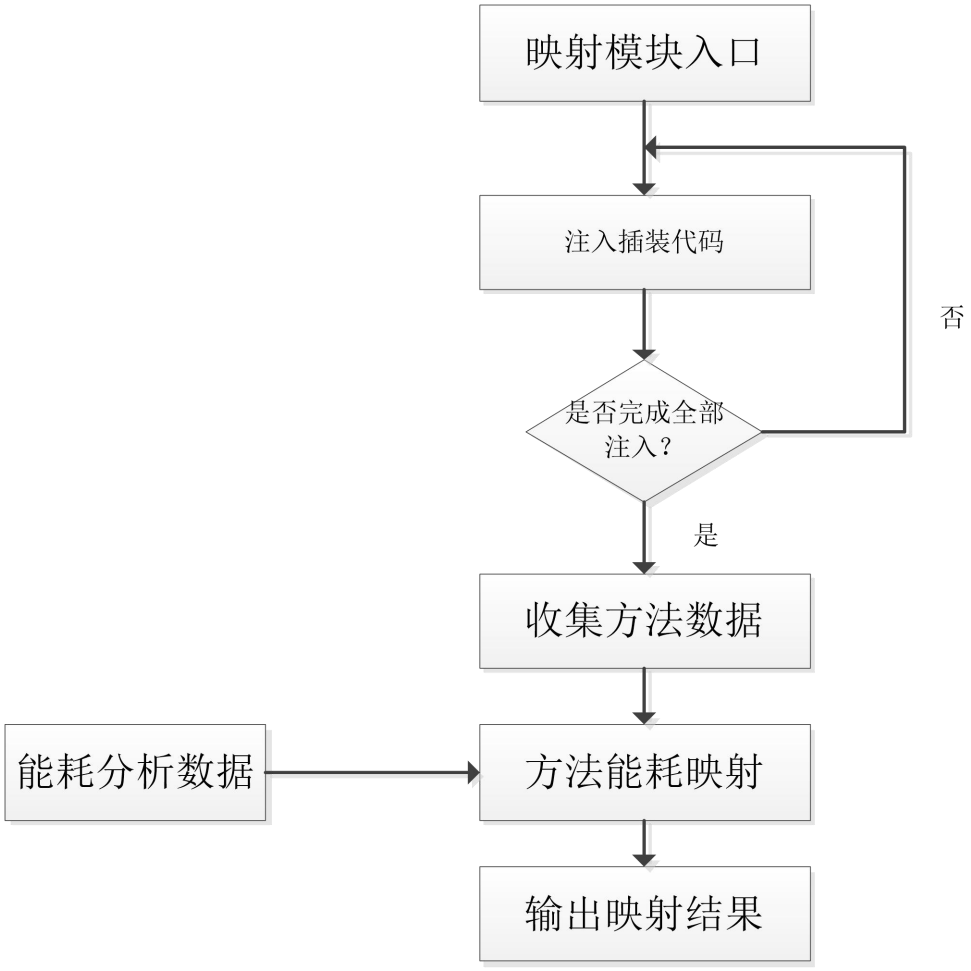

图 4.5　字节码插装流程图

注入可以在运行时进行，插装代理在第一次加载类时注入额外的代码；也可以在离线时使用特殊工具将代码注入到程序的.class 文件中。

因此，第一种方法是一个在运行时测量字节代码并估计能源消耗的代理，而第二种方法由两个工具组成：一个离线测量字节代码的应用程序和一个在运行时估计能源消耗的代理。两个版本注入相同的代码并提供相同的计算。第二种方法的主要优点是它减少了开销（因为插装是在脱机状态下完成的），并且它可以对所有类进行插装，包括在运行时使用不同的类加载器加载的类（在这种情况下，实现的第一种方法无法对它们进行插装）。代码对所有方法执行类似的任务（例

如收集方法名、基于注入的计数器的执行时间以及它在调用树中的位置）时，额外的代码会产生固定的开销。在方法开始处注入的代码收集信息，如方法的全名、方法执行的时间戳和方法调用树中的深度（用于检测调用的子方法）。这个信息有助于确认能量是在哪里消耗的（例如，一个方法消耗的能量不包括它的子方法消耗的能量）。深度从 0 开始，在每个方法入口递增，在方法出口递减。因此，在执行序列中，这个值表示当前监视的方法在调用树中的位置。

在方法末尾注入的代码会收集方法末尾的时间戳，并考虑到调用树在一个级别上的恢复（当一个方法结束时，会回到它的父方法或主方法）。这是因为 Java 在其 Java 堆栈中使用堆栈帧。堆栈帧包含一个 Java 方法调用的状态。当线程调用一个方法时，Java 虚拟机将一个新框架推入该线程的 Java 堆栈。当方法完成时，虚拟机弹出并丢弃该方法的框架[13]。

在方法退出时，我们计算方法的执行时间，不包括它的子方法的执行时间。当一个方法存在时，我们检查它是否有父类（深度-1），然后将它自己的执行时间添加到它父类的 methodinfo 对象（该对象包含关于该方法的统计信息）。因此，每个方法的执行时间是通过方法的总执行时间（进入和退出之间）减去所有子方法的执行时间来计算的。

网络信息是通过将套接字方法的所有方法调用路由到一个自定义的实现，在这个实现中，通过添加计数器来计算每个方法发送和接收的字节数，重写方法 getinputstream()和 getoutputstream()，以监视读写到套接字的字节数。然后，该信息与调用 getInputStream()或 getOutputStream()方法的方法名相关联，以获得方法读取/写入的字节数。

在每个监视器周期或程序执行结束时，映射模块会周期性地处理每次方法调用时收集的数据。它应用能耗模型来提供终端上的每个方法消耗的能量记录或保存在文件中的能量记录。

（2）统计抽样实现

能耗映射模型的统计抽样版本从 JVM 收集运行方法的有关信息，并将它们与我们的能量模型关联起来。该版本遵循的采样策略如图 4.6 所示。

我们首先遵循一个循环调用的方法，方法如下：

① 在应用程序监视周期（500 毫秒或 1 秒）中，使用在能耗模型中实现的方法监视整个应用程序的能源消耗。这为我们提供了应用程序的能源消耗，我们将使用它作为所有执行方法的总能源消耗。

② 下一步是根据应用程序的执行情况在应用程序的方法之间分配应用程序的能量消耗。具体来说，我们通过查看 JVM 的堆栈跟踪来检测当前正在执行的

方法。我们经常检测和计算每个方法在堆栈顶部的次数，通常是每个 10 毫秒（例如，代码监视周期）。

③ 然后，我们将这些统计数据与线程的 CPU 时间（从 JVM 收集的）关联起来，用来估计方法的能源消耗。

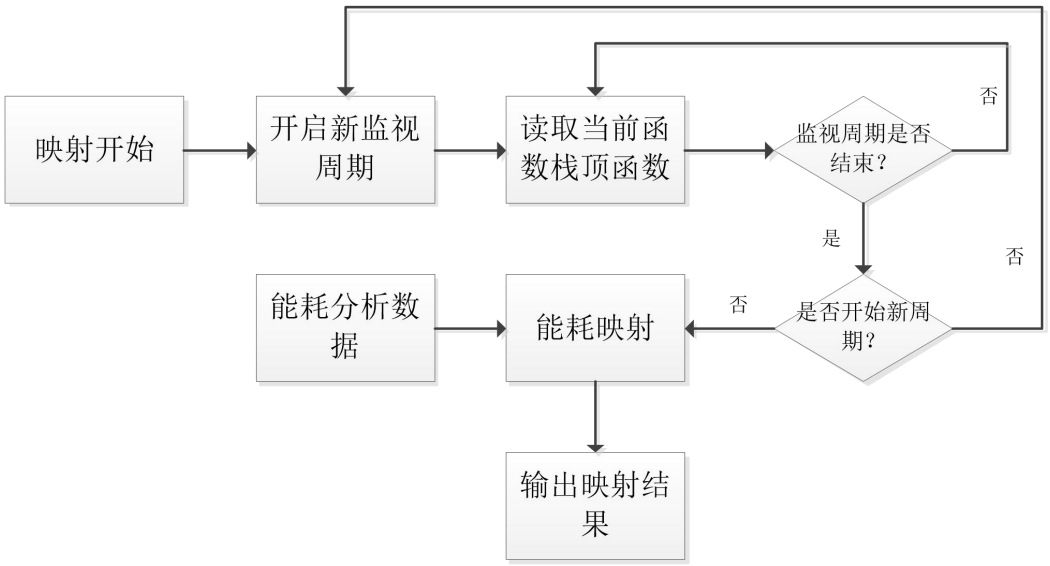

图 4.6 统计抽样流程图

例如，红色和蓝色两种方法执行 10 秒，应用程序周期为 1 秒，代码周期为 10 毫秒。这些方法中的每一个都有不同的执行时间和 CPU 利用率，因此这两个方法都被相应地调度和执行（例如，蓝色方法等待网络应答，因此 JVM 在等待期间执行红色方法）。在循环过程中，蓝色方法在堆栈顶部检测到 10 次，而红色方法检测到 12 次。因此在应用程序监视周期中，蓝色的方法可以获得 45%的能耗，而红色方法获得 55%的能耗（参见图 4.7）。

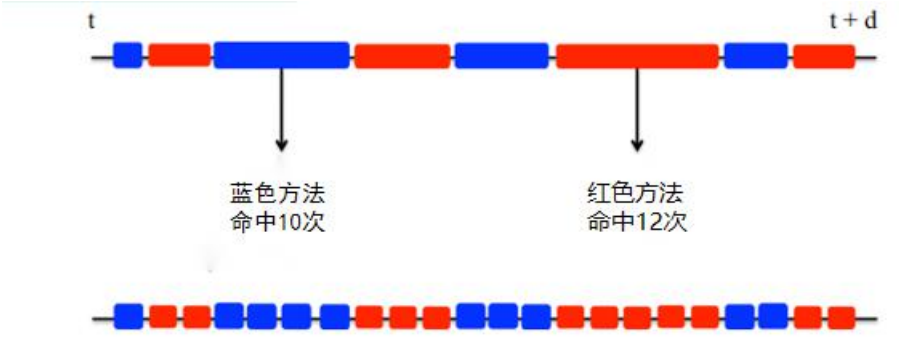

图 4.7 统计抽样示例图

网络能耗通过检测和计数调用 Java JDK 的负责网络的方法（像 java.net 这样的方法），并应用于类似在前面的步骤中所述的能量分布处理的所有方法调用 Java JDK 网络的方法。

4.3 能耗分析初始化实现

能耗分析初始化主要用来计算代码程序中各个方法的结构特征并记录对应代码位置，为代码文本处理工具，具体步骤如下：

（1）读取目标程序代码，直到读取到类中第一个方法，进入步骤 2。

（2）分析读取到的方法的效代码行数、构件个数、算法复杂度等方法内部独立结构特征，并记录该方法所在的包与类、调用深度、耦合程度。

（3）将方法置入目标程序代码的方法生成树中，根据其位置更新其调用深度，并对其余所有已读取代码的调用深度进行更新。

（4）分析方法的耦合程度。完成后重复步骤 1，直至读取完所有代码，生成文件并退出。

4.4 能耗预测实现

4.4.1 自适应线性网络实现

使用自适应线性网络实现线性拟合的具体流程如下：

（1）对样本程序所有方法的代码结构特征量（有效代码行数、构件个数、算法复杂度、调用深度、耦合程度）分别进行度量。

（2）对软件特征量的度量值进行标准化的预处理，并将其拉伸为一维向量，作为线性分析的输入值。

（3）运行样本程序，并用能耗评估模型计算样本程序的各个方法的能耗值，作为线性分析的输出值。

（4）输入样本程序代码结构特征量和能耗值对线性分析网络进行训练，并利用 K 折交叉检验等优化模型。

（5）输入目标程序的特征量到训练好的线性回归模型中，得到能耗预测值，再与目标程序的实际能耗值进行对比，验证建模方法的有效性。

其中预处理是指，如果将取值范围差异很大的数据输入到神经网络很对预测产生很大问题，网络可能会自适应这种取值范围不同，但是学习肯定会变得更困难。对应这样的数据，通常对每个特征进行标准化。方法即，减去每个具体特征的测量平均值，再将其差值除以标准差。这样采用计算方法得到的数值特征误差平均值一般为 0，标准差的特征平均值一般为 1。K 折优化算法的具体做法是，将数据划分为 K 个相同的子分区（通常 K 是 4 或者 5）。实例化 K 个相同模型，

每个模型在 K-1 个分区上训练，在剩下一个分区上验证评估。模型的验证分数等于 K 个验证分数的平均值。其余如代价函数等前面章节已有说明，这里不再赘述。

4.4.2 BP 神经网络实现

使用 BP 神经网络实现非线性拟合的具体流程如下：

（1）对样本程序所有方法的代码结构特征量（有效代码行数、构件个数、算法复杂度、调用深度、耦合程度）分别进行度量。

（2）对软件特征量的度量值进行标准化预处理，处理后的值作为 BP 神经网络的输入值。

（3）运行样本程序，并用能耗评估模型计算样本程序的各个方法的能耗值，作为 BP 神经网络的输出值。

（4）确定 BP 神经网络的具体结构，包括隐层数、隐层节点数、隐层传递函数、输出层传递函数等。

（5）输入样本程序代码结构特征量和能耗值对 BP 神经网络进行训练，通过训练确定 BP 神经网络隐层的权值和阈值。

（6）输入目标程序的特征量到训练好的 BP 神经网络中，得到能耗预测值，再与目标程序的实际能耗值进行对比，验证建模方法的有效性。

在获得有效的代码结构特征度量值后，需要进一步考虑的因素包括神经网络学习速率、逼近误差、收敛速度、内存占用情况等来确定和优化 BP 神经网络的结构。

隐含层数的确定：过多的隐含层数会导致神经网络复杂化，加模型训练时间和过拟合的倾向，导致预测结果误差增大。实验证明，单隐层的 BP 神经网络可以逼近任何在闭区间内的连续函数，一个 3 层的 BP 网络可完成任意的 n 维到 m 维的映射。因此，隐层数确定为 1。

隐层节点数的确定：BP 神经网络中，隐层节点数对 BP 网络性能影响巨大，但目前理论上还缺乏科学的和普遍的方法用于隐层节点数的确定。一般采用以下经验公式确定隐层节点数

$$n_1 = \sqrt{n+m} + a \tag{5.1}$$

其中，n 为输入层节点数，m 为输出层节点数，a 为 1~10 的常数。对应到体系结构级能耗模型中，输入层包括有效代码行数、构件个数、算法复杂度、调用深度、耦合程度 5 个输入量，因此，输入层节点数为 5，输出层为平均功耗 p，输出层节点数为 1。因此，n_1 的取值范围为 3~12。

通过多组的实验我们可以直观地发现，在隐层的节点收敛函数为 11 时，其节点收敛的速度和逼近的误差都能同时达到非常令人感到满意的节点收敛效果。因此，bp 神经训练函数的隐层节点收敛函数值设定为 11。

4.5 能耗评价与优化方案实现

4.5.1 能耗评价实现

具体步骤如下：

（1）根据预测模型预测的代码能耗与初始化模块生成的方法信息与代码生成树计算各个方法累积产生的程序的总能耗。

（2）累计各个方法的有效行数，并与程序总能耗一同分析（如与有效代码总相同的其他程序比较，方法不一，因高能耗界定标准不同）程序是否存在高能耗隐患。

（3）如存在高能耗隐患，计算各个方法产生的累计能耗在总能耗中的占比，找出高能耗代码。

（4）将高能耗代码相关信息（包含其他模块传递的信息）本地储存。

4.5.2 优化方案实现

此类方案相关的研究十分深入，对代码优化从指令级、语句级、模块级分别对代码的优化给出了建议并给出了具体工具实现，本文就不再赘述，本文中采取的优化方法即为文献[14]中所叙述的，具体步骤如下：

（1）根据评价模型对高能耗代码的定位从结构特征分析其高能耗成因。

（2）根据成因找寻对应优化方法进行简单优化，并通过预测模型预测优化后的方法能耗与优化前方法能耗比较。

（3）如优化有成效则记录并本地保存，如优化未见成效则尝试其他办法。

（4）用户读取优化方案的意见并做出相应更改。

（5）利用预测模型与评估模型对优化后的程序重新进行预测与评估，并与优化前比较。

4.6 能耗工具整体实现

最终，将上述全部模块功能实现并整合后，工具得以实现，图 4.8 为工具界面图，操作方式为从控制台输入命令来选择功能，下面将分别详细介绍：

```
================================================
=========欢迎使用Javaprofiler能耗分析工具 =========
===============请选择您要进行的操作 ===============
=          1 监测评估能耗                        =
=          2 训练模型                            =
=          3 预测代码能耗                        =
=          4 高能耗定位与优化                    =
=          5 退出系统                            =
================================================
请输入您的选择
```

图 4.8　工具主界面图

（1）监测评估能耗

```
请输入您的选择
1
请选择监测方式：1.按进程号监测 2.自启动监测 3.返回上一级
1
请输入进程号
12708
开始监测
监测完成，能耗数据文件已生成
```

图 4.9　工具监测评估能耗操作图

可选择按进程号监测或自启动监测，按进程号监测则需先启动待检测程序获取进程号并输入工具中以进行监测，自启动监测则需将待监测程序放入 monitor 文件夹中，工具可自行启动该程序并进行能耗监测，监测结束后将能耗文件存入 monitor 文件夹下。

（2）训练模型

```
请输入您的选择
2
请选择模型：1.线性 2.非线性 3.返回上一级
2
请输入训练集名称，并将其放入train文件夹中
test
开始训练
训练完成，非线性模型已生成
```

图 4.10 工具训练模型操作图

可选择训练线性或非线性模型，需要输入对应的训练集名称并将其放入 train 文件夹下，工具则可读取相应文件并作为调用相应训练模型的子程序的参数，等待子程序训练完成，返回结果，新训练成功的模型统一命名为 new 并生成在 train 文件夹中，如需保存可更改模型名称。

（3）预测代码能耗

```
请输入您的选择
3
将预测集放入forecast文件夹中并命名为test
请输入选择的模型文件名称：
line
预测完成
预计总能耗为102J，各方法能耗已生成在预测文件中
```

图 4.11　工具预测代码能耗操作图

可选择之前训练完成的模型，需要预测集其放入 forecast 文件夹下，工具则可读取相应文件并调用模型预测能耗数据，等待预测完成返回结果保存在 forecast 文件夹下。

（4）高能耗定位与优化

可将前面过程的能耗数据文档放入 optimization 文件夹中并在工具中选择对其进行高能耗分析，返回分析结果，显示定位到的高能耗方法，并给出优化建议，详细优化步骤需查询优化方案文档，如图 4.12 所示：

```
请输入您的选择
4
请输入选择的高能耗分析文件名称并将其optimization放入文件夹中：
test
定位分析完成，以下为优化方案
orderdispatch  耦合程度高，建议拆分
someorderupdate  循环展开，降低圈复杂度
详细优化方案参考优化方案
```

图 4.12　工具高能耗定位优化操作图

（5） 系统退出

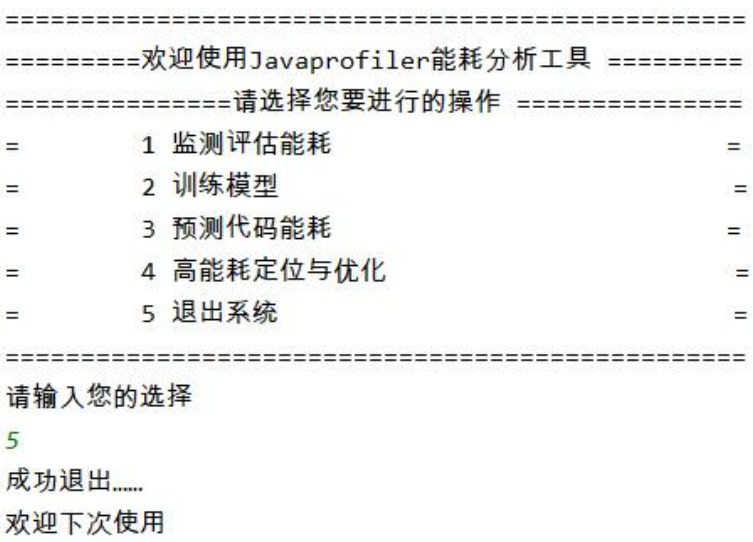

图 4.13　工具退出界面图

退出工具，关闭所有正在运行的子进程。

5 实验验证与实验分析

5.1 能耗模型验证

5.1.1 能耗评估模型验证

（1）CPU 模型验证

首先验证能耗评估模型提供的结果的准确性。将能耗评估模型提供的功率值与使用功率计的计算机的实际功耗进行了比较。注意，由于功率计和我们的库之间的同步时间延迟，值在监视开始时会移动几秒钟。这些值都进行了标准化，以便观察 CPU 功耗的趋势。

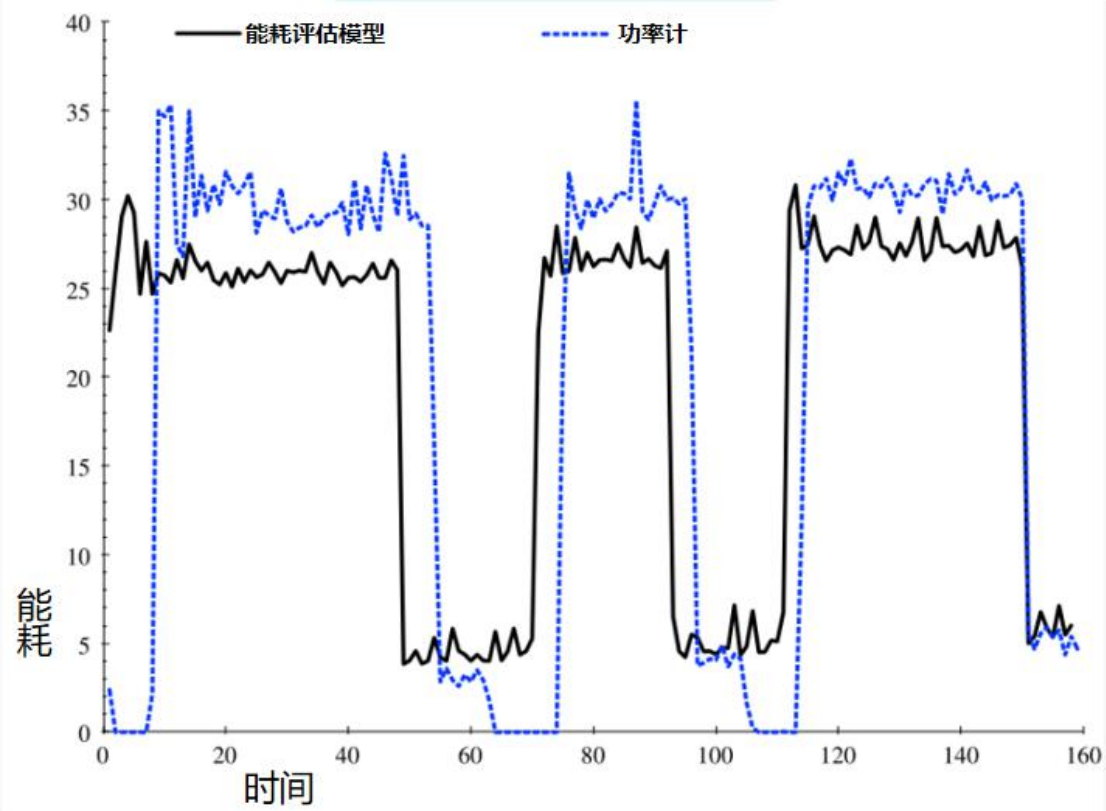

图 5.1CPU 能耗监测图

如图 5.1 所示，功率计监视整个计算机的能源消耗，包括它的所有主要硬件和系统组件、所有的软件应用程序和所有它的操作系统。因为功率计在空闲时间也会统计能量消耗，为了将值与能耗评估模型进行比较，实际测量时从运行实验时监控的能量中减去空闲时间的能量消耗。具体来说，在空闲时间，使用功率计（例如 E_idleMeter）测量计算机消耗的能量。然后进行实验，再次测量计算机

的能量（例如，E_totalMeter）。这两个值的区别在于应用程序消耗的能量。但是，这个值对应于应用程序所强调的是所有硬件组件的能量（例如 CPU、内存等）。因为模型只监视选择的硬件组件，比如 CPU 或网卡，所以决定根据功率计与我们的能耗评估模型的偏移量（例如，E_offset = E_totalMeter−E_appLibrary）对功率计值进行规范化。因此，实现验证了能耗评估模型提供的值小于 E_appMeter，并且其可能代表了一个更好的预测值，可以用功率计的值作为规范化偏移量加入到模型中。最后，本文计算了功率计的值与模型的值之间的差值的平均值并为功率计的每个测量值减去这个计算平均值。结果显示，模型估计的值与实际功耗之间存在微小的差异。在测试场景中，误差范围估计为标准化值和平均值的 0.5%（当同步延迟时，单个测量的平均误差高达 8%）。在压力测试中，误差增长到近 3%（个别测量的平均误差为 13%）。因此，可以合理地认为，使用以能耗评估模型为中心的方法提供的值足够精确。

（2）网络模型验证

使用 iperf11 运行一个网络压力测试，并在我们的主机配置上测量 Iperf 的 CPU 服务器的功耗。从一个分布式客户机向我们的主机服务器发送两组各为 100MB 的 TCP 数据包。使用了 Iperf 的默认设置，其中它的 CPU 服务器也按周期（每秒）执行。结果显示网络消耗在 0.017 W 左右，而 CPU 消耗在 0.9 W 左右。这一观察结果与文献[15]相关。因此，本文主要概述了 CPU 实验的结果。

5.1.2 能耗映射模型

两个版本的能耗映射模型，字节码检测和统计采样，都使用能耗评估模型作为一个应用程序库。进行两组实验：首先，我们测量能耗，并将其与软件分析器的能耗进行比较，其次，通过比较在 100% CPU 下运行的 CPU 密集型应用程序的能量变化和 CPU 时间变化以及与另一个软件分析器的比较，来评估准确性。根据其他相关研究，对于一直以 100%的 CPU 利用率运行的软件，它们的能源消耗和执行时间之间存在线性关系。

由于没有其他的软件分析器提供软件代码的能耗，我们通过比较一些进程分析工具提供的能耗和 CPU 时间来验证我们方法的准确性。因此，我们使用相同的 CPU 密集型应用程序，即递归 Java 版本的河内塔程序，以证明能耗映射模型提供的结果是准确的。我们比较了能耗映射模型提供的能量信息和估计的 CPU 时间的方法。TowerOfHanoi.moveDisk 方法耗了 83.34%的 CPU 和它的能量，而 TowerOfHanoi.solveHanoi 方法的消耗能量为 16.58%。最后，主方法消耗 0.06%的能量。

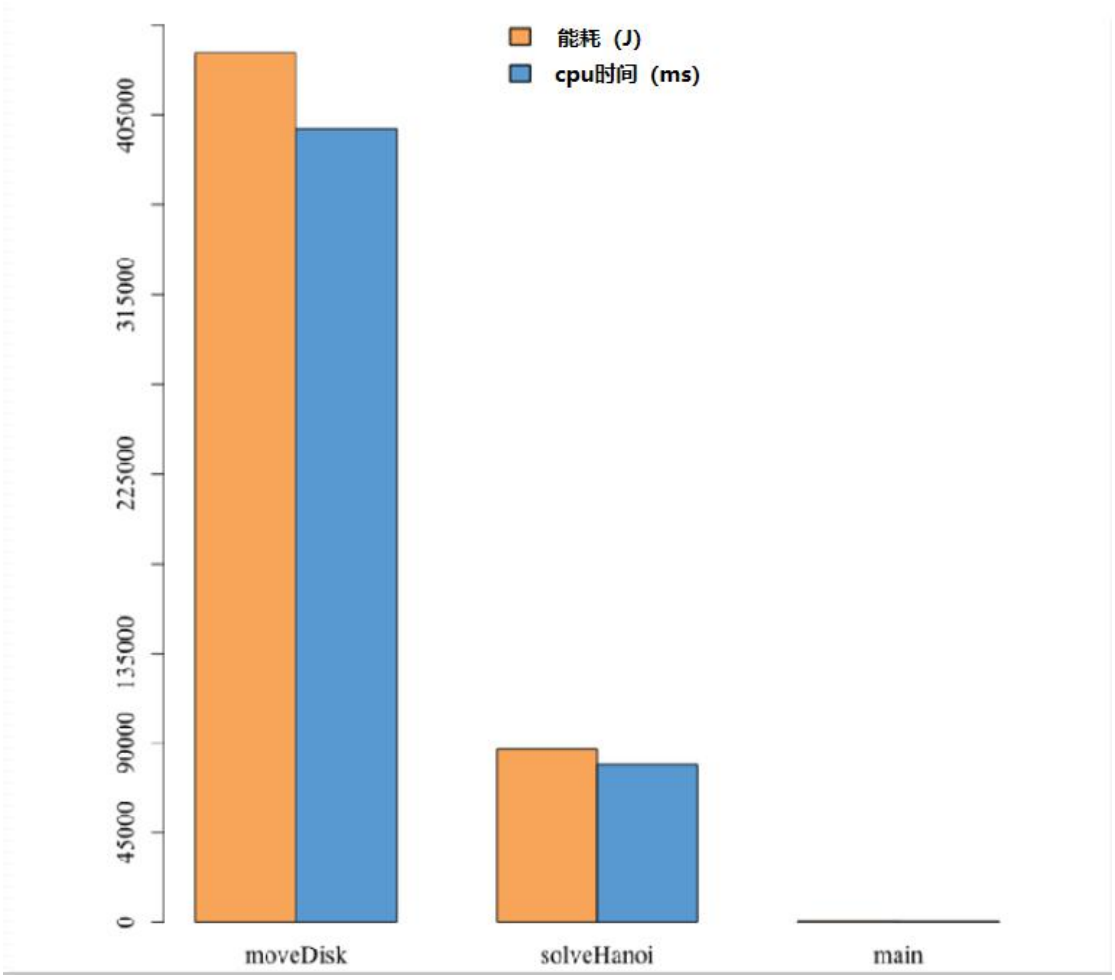

图 5.2 汉诺塔程序递归版本的能耗与 CPU 时间对比图

图 5.2 中的结果显示了 CPU 时间和能量之间的相似匹配，这是和所预期的一致，因为在这种情况下，时间和能量是线性相关的。另一方面，抽样版本不使用 CPU 时间来估计软件代码的能源消耗。因此，决定将其与 HPROF profiler 进行比较，HPROF profiler 是一种软件 CPU 分析工具，它也使用统计抽样来估计软件代码的 CPU 使用情况。Java 2 平台标准版（J2SE）在缺省情况下提供 HPROF，作为命令行工具。此工具估计在 JVM 中执行的所有方法的 CPU 利用率百分比。与此相反，能耗映射模型可以估计所有方法的能量消耗，但是也可以将这种估计过滤为一些方法的选择（例如限制估计为河内塔的方法，同时排除对 Java JDK 方法的调用）。我们比较了能耗映射模型提供的能量消耗和 HPROF 提供的输出信息。HPROF 报告的结果显示 java.io.FileOutputStream.riteBytes 方法在程序执行期间使用了 97.33%的 CPU。该方法的能耗为 96.05%，与 HPROF 相比，差异为 1.3%。

然而，能耗映射模型也可以过滤掉相应的方法，当排除 jdk 的方法时，结果显示 TowersOfHanoi.moveDisk 方法消耗了 99.92%的能量。moveDisk 方法消耗了 99.92% 的能量。这是因为 TowersOfHanoi.moveDisk 调用 java.io.FileOutputStream.writeBytes 方法（以及其他方法，如 java.io.BufferedWriter.write 或 java.io.Writer.write）来将程序的结果写入文件。此外，TowersOfHanoi.moveDisk 本身的净能耗为 0.73%（HPROF 报告仅此方法的能耗为 0.13%）。

5.2 预测模型验证

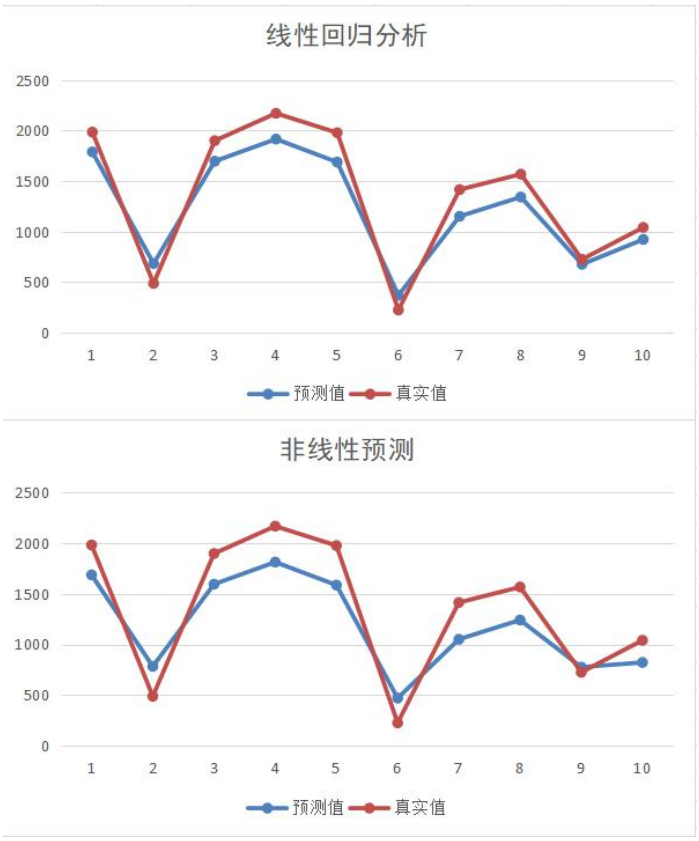

图 5.3 高相似度抽样对比图

为了验证两种能耗预测模型的有效性与差异性，本文采用 3 组测试方案分别对模型进行验证，这些测试方案中包含了许多较为常用的 Java 程序，包含了各个平台的应用程序，其中第一组为与两个模型训练集相似的 CPU 密集型 Java 在 PC 端上的程序，第二组为与实验所在平台存在较大差异性的 Android 端应用程序，第三组为在样本库中各随机抽取数据进行测试（所有测试程序均在样本库

中，包含各个平台的常用程序）。本文中预测模型所使用的能耗模型来自上述能耗模型。

（1） 第一组验证

测试结果如图 5.3，可以看出，在对相似度较高的程序（结构或框架相似）进行静态预测分析时，线性回归分析明显优于非线性预测，线性回归分析的最大误差为 18.3%，最小误差为 6.8%，平均误差为 16.7%，非线性预测的最大误差为 26.4%，最小误差为 7.3%，平均误差为 20.3%。这证明了当需要预测的程序有很多结构相似的程序作为样本时，应选择线性回归分析的方法。

（2） 第二组验证

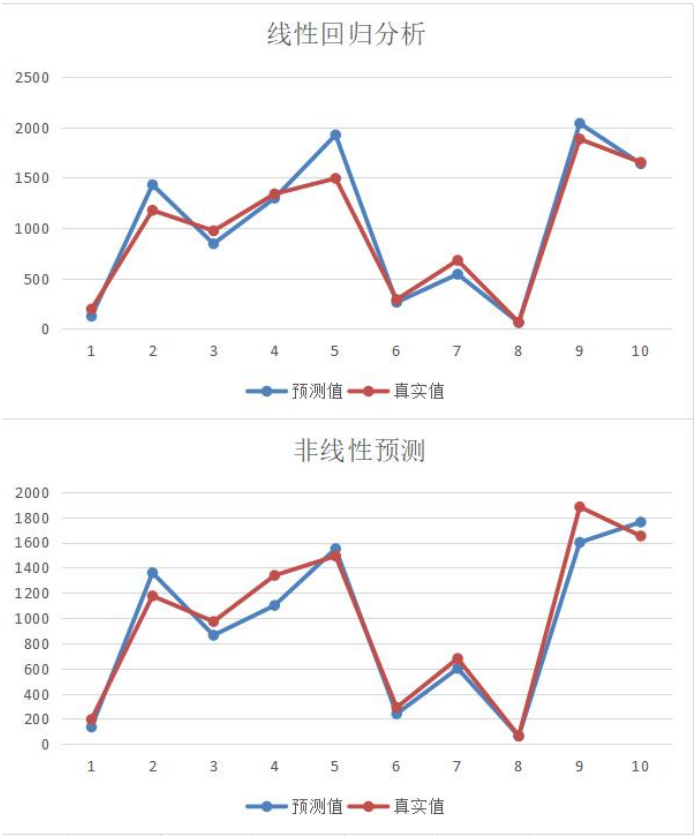

图 5.4 跨平台抽样对比图

如图 5.4，可以看出，在对跨平台的程序进行静态预测分析时，线性回归分析的结果与上一组差异明显，非线性回归预测同样与前一组存在差异但差异较小，线性回归分析的最大误差为 41.7%，最小误差为 7.2%，平均误差为 23.6%，非线性预测的最大误差为 33%，最小误差为 8.5%，平均误差为 21.2% 。

（3） 第三组验证

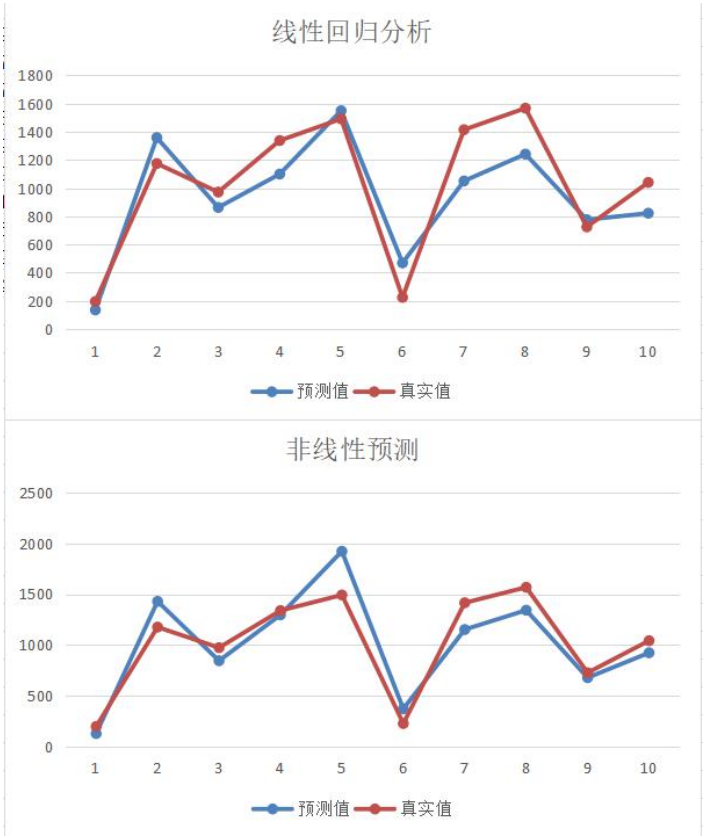

图 5.5　随机抽样对比图

抽取了一组对比实验，可以看出，在对未知平台的未知程序进行静态预测分析时，线性回归分析的预测结果不稳定，方差与极差均高于非线性预测。

综上所述，对已经训练好的模型来说，对与其训练集样本框架类似的样本进行预测时，应使用线性回归分析，对跨平台的未知程序进行预测时应选用非线性预测。同时也证明上述模型选取的特征量与能耗存在相应关系，利用方法层面预测代码能耗的方案是有效的。

5.3　能耗定位与优化方案验证

抽取了一份样例程序进行验证，该样例程序的总能耗与能耗定位的占比图如图所示，为了清晰表示图中方法名全部使用字母代替：

从图 5.6 中可以看出，方法 H 在总能耗中占比高达 25%，有效代码行数占比也远高于其他方法，方法 B 在总能耗中的占比显著高于在有效代码行数中的占比，两者皆为高能耗方法。下面利用优化方案对其进行优化：首先对于方法 H，分析其代码结构特征（特征值为：有效代码行数 367、构件个数 5、算法复杂度 2.1、调用深度 3、耦合程度 0.7），找出能耗原因。根据分析，发现是由于在较高有效代码行数情况下其调用深度较低情况下构建个数于耦合程度高带来的能耗影

响，遂进行降低耦合程度的优化方案，将方法拆分降低耦合程度（方法拆分需要开发者对于具体功能点进行划分，这里只能给出建议）。

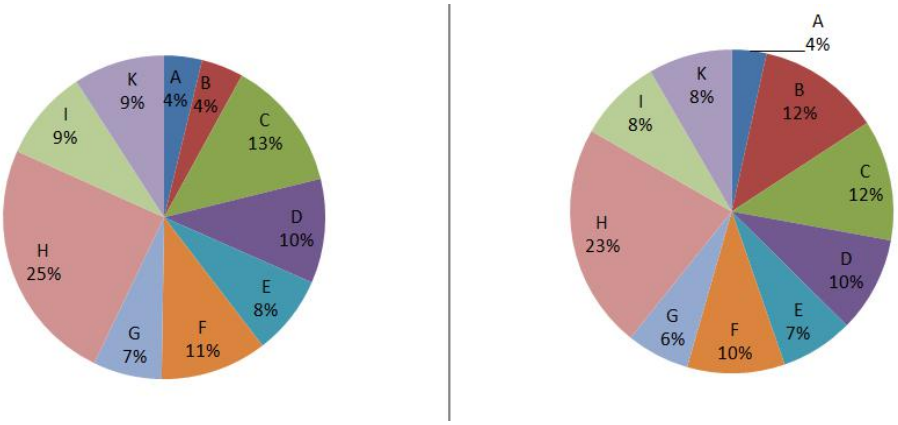

图 5.6　优化前代码行数占比图（左）与能耗占比图（右）

对于方法 B，分析其代码结构特征（特征值为：有效代码行数 91、构件个数 3、算法复杂度 3.2、调用深度 5、耦合程度 0.35），找出能耗原因。根据分析，发现是由于其调用深度较高情况下高算法复杂度带来的能耗影响，遂进行降低算法复杂度能耗的优化方案，将循环展开。循环展开通过多次展开“各语句数据无关”的循环体来增加循环语句，减少循环次数。循环展开旨在增加循环体指令级并行度，这种变换减少了循环控制语句执行的次数，为具有多个功能单元的处理器 提供指令级并行，也有利于指令流水线的调度。

图 5.7 为优化后的能耗定位图，可以看出方法 H 与 B 的能耗（计算 Y 拆分后的方法能耗和）分别降低了，证明了本文的方法是有效的。

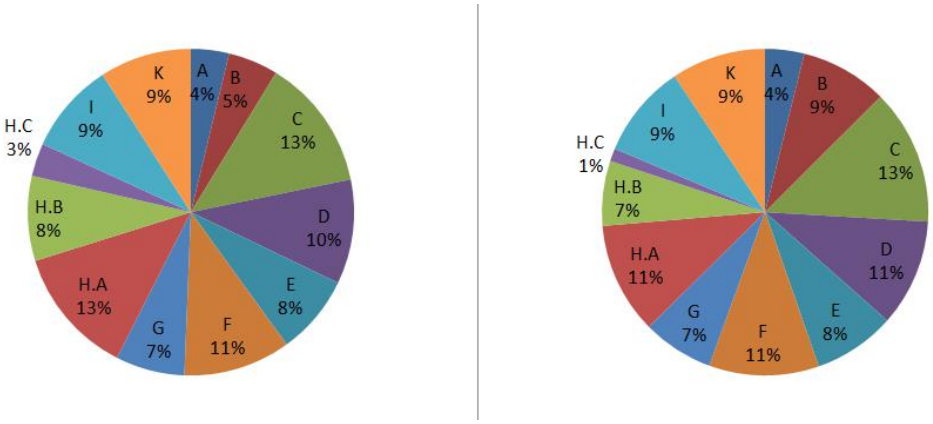

图 5.7　优化后代码行数占比图（左）与能耗占比图（右）

6 总结与展望

6.1 总结

近年来随着信息技术及其相关产业的高速发展，IT 相关的设备的使用规模不断地再扩大，能源生产的功耗急剧增加给环境带来了沉重的压力。高能源消耗成为了制约 IT 相关产业发展的主要原因。在软件设计开发过程中，除了传统意义上的性能与可扩展性两个度量指标外，软件的能耗成为了新的度量指标。本文提出了将软件产生的能耗对应到其代码的方法上，利用分析方法级别代码结构特征以分析其能耗规律的思路并以此设计了能耗评价方法。理论模型所涉及的主要工作如下：定义了能耗评估模型，包括能耗分析模型和能耗映射模型，为能耗评价提供了理论依据。提出了代码结构特征对于能耗分析的理论基础，提出以神经网络分析模型作为能耗分析的切入点。

设计了能耗预测模型，给出了不同条件下的应用方案。最后，对实际环境中的能耗模型、预测模型和方法进行了比较和验证，证明了所提出方法的可行性和正确性。最后，本文的结论如下：

（1） 在有效的能耗评估模型下，可在操作系统层面下收集相关运行信息来统计软件代码的能耗，误差精度在 3%左右，是比较精确的方法。

（2）通过字节码插装或统计抽样的方式定位代码能耗在去掉方法本身额外的开销后对定位的代码能耗的误差在 5%左右，误差主要来自统计间隔或采样间隔。

（3） 线性回归分析模型对于代码能耗的预测是行之有效的，在结构类似的前提下预测误差精度最低可达 10%，而非线性回归预测适合分析预测结构不相似的样本，误差精度在 20%左右。

（4）可通过计算各个方法产生的能耗在总能耗的占比定位高能耗代码并对其进行优化。

最终，本文实现了代码能耗分析工具 Java-profiler 的设计与实现。

6.2 展望

在软件开发过程中，如何降低能耗已经成为了重要指标。本文实现了在软件开发的设计阶段分析与评估相关的能耗特征，可以提早发现代码存在的潜在能耗问题，有助于提高开发效率、降低开发成本，同时实现了能耗优化，符合目前倡导的节约型经济和低碳经济政策。总之，本文具有先导性、前沿性和实用性。

本文研究仍处于基础阶段，对于模块间的更深层次的关系对能耗的影响没有深入考虑，没有能够建议一个统一高效的模型。所以如何提高实现层面的适用性，设计一站式的、更自动化与全面的软件能耗评估方法，是今后的思考和研究方向。

参 考 文 献

[1] 刘雪冰. 面向代码结构的软件模块能耗评估方法[D]. 沈阳:东北大学软件学院,2014.

[2] Tan TK, Raghunathan A, Lakishminarayana G, et al. Proceedings of the 38th annual Design Automation Conference, New York, 2001[C]. New York: Association for Computing Machinery,2001.

[3]Tan TK, Raghunathan AK, Jha NK. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany, 2003[C]. Munich: Publisher unknown,2003.

[4] Lee I, Philippou A, Sokolsky O. Process algebraic modelling and analysis of power-aware real-time systems[J]. Computing and Control Engineering Journal, 2002, 13(4):180-188.

[5] Senn E , Laurent J , Juin E , et al.Proceedings of the Forum on Specification, Verification and Design Languages, Stuttgart, 2008[C]. Stuttgart: Publisher unknown,2008.

[6] Zhang TT, Wu X, Li CD et al. On energy-consumption analysis and evaluation for component-based embedded system with CSP[J]. Chinese Journal of Computers, 2009,32(9):1-8.

[7] Chen LQ, Shao ZQ, Fan GS. Energy consumption modeling and analysis for distributed real-time and embedded systems [J]. Journal of East China University of Science and Technology (Natural Science Edition), 2009,35(2):250-255.

[8] Zhao X, Guo Y, Lei ZY, et al. Estimation and analysis of embedded operating system energy consumption[J]. tien tzu hsueh paoacta electronica sinica, 2008,36(2):209-215.

[9] Noureddine A , Rouvoy R , Seinturier L . Monitoring energy hotspots in software Energy profiling of software code[J]. Automated Software Engineering, 2015,22(3):291-332.

[10] Feeney, L.M., Nilsson, M. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Society, Anchorage, USA, 2001[C]. Anchorage: Conference on Computer Communications,2001.

[11] 阳馨.嵌入式软件体系结构级能耗建模方法[J].电子测试,2014(S1):30-32.

[12] 刘啸滨, 郭兵, 沈艳,等. 嵌入式软件体系结构级能耗建模方法[J]. 软件学报, 2012(02):58-67.

[13] Venners, Bill. Inside the Java Virtual Machine[M]. New York: McGraw-Hill Companies,2000.

[14] 宋杰, 孙宗哲, 李甜甜,等. 面向代码的软件能耗优化研究进展[J]. 计算机学报, 2016, 039(011):2270-2290.

[15] Rivoire S, Shah MA, Ranganathan P, et al.Proceedings of the ACM SIGMOD International Conference on Management of Data, New York, USA2007[C]. New York: Association for Computing Machinery,2007.

修改记录

第一次修改记录：

第 23 页图 4.5，**修改前：**判断条件：是->继续插装

修改后：判断条件：否->继续插装

第二次修改记录：

第 35 页图 5.3，**修改前：**分线性预测

修改后：非线性预测

第三次修改记录：

第 31 页图 4.13 图名，**修改前：**工具高能耗定位优化操作图

修改后：工具退出界面图

第四次修改记录：

（5） 毕业设计（论文）外文翻译修改记录：无

（6） 毕业设计（论文）正式检测重复比：17.9%

记录人（签字）：唐智强

指导教师（签字）：

致　　谢

从论文选题的开始到如今论文的基本完成，经历了数个月的钻研与探索，紧张而又充实的本科毕业设计终于落下了帷幕。在这次本科毕业设计的完成过程中，学到的不仅仅是专业相关的技术知识和论文书写的严谨态度，更多的是对于事物孜孜不倦的探求精神以及对工作时间的合理规划，这些都将成为我这一生中最宝贵的财富。

值论文即将完成之际，我想要对所有在完成过程中为我提供过帮助的家人、同学、我的导师以及其他老师表示深深的感谢！

感谢我的导师——侯刚老师。从毕设选题开始，到后来工具的设计与实现，再到后来论文的撰写与修改，侯刚老师都给予了我很大帮助。前期多次探讨帮助我找寻设计思路，后期多次询问进度并就可能出现的问题给予纠正和引导，及时督促我完成论文并对论文中出现的问题给出修改意见。在侯刚老师的悉心指导下，这篇论文才能够顺利的完成。

感谢我的家人，他们在是我背后默默的支持者与鼓励者。

感谢各位专家、老师在百忙之中审阅我的论文并提出宝贵意见。

感谢大连理工大学软件学院对我的栽培，四年的宝贵学习时光是我终生难忘的经历。