Cooper由需求到架构

[一、 说明业务 3](#_Toc356992049)

[1.1、 识别业务问题 3](#_Toc356992050)

[1.2、 定义软件系统 4](#_Toc356992051)

[二、 分析需求 4](#_Toc356992052)

[2.1、 Cooper要解决的问题 4](#_Toc356992053)

[2.2、 Cooper需要做到的 4](#_Toc356992054)

[2.3、 Cooper的产品特性 5](#_Toc356992055)

[2.4、 Cooper的客户和用户 5](#_Toc356992056)

[2.5、 Cooper的功能性需求 6](#_Toc356992057)

[2.6、 Cooper的非功能性需求 6](#_Toc356992058)

[2.7、 需求中的核心概念 7](#_Toc356992059)

[2.8、 Cooper的关键架构需求 7](#_Toc356992060)

[三、 设计架构 8](#_Toc356992061)

[3.1、 Cooper的架构目标 8](#_Toc356992062)

[3.2、 Cooper的架构策略 8](#_Toc356992063)

[3.3、 确定系统级别的关键交互 9](#_Toc356992064)

[3.4、 规划系统的高层组织 9](#_Toc356992065)

[3.4.1、 识别分析类 9](#_Toc356992066)

[3.4.2、 初步划分分析包 11](#_Toc356992067)

[3.4.3、 考虑分析机制 13](#_Toc356992068)

[3.4.4、 精化分析包 14](#_Toc356992069)

[3.4.5、 描述用例实现 15](#_Toc356992070)

[3.5、 演进系统的高层组织并形成组件模型 18](#_Toc356992071)

[3.5.1、 初始化组件模型 18](#_Toc356992072)

[3.5.2、 确定设计机制 20](#_Toc356992073)

[3.5.3、 精化组件模型 22](#_Toc356992074)

[3.5.4、 描述用例实现 23](#_Toc356992075)

[3.6、 考虑系统的运行时模型 25](#_Toc356992076)

[3.6.1、 单机模式 25](#_Toc356992077)

[3.6.2、 客户端模式 25](#_Toc356992078)

[3.6.3、 浏览器模式 26](#_Toc356992079)

[3.6.4、 插件模式 26](#_Toc356992080)

[3.7、 规划系统的部署模型 26](#_Toc356992081)

[3.7.1、 单机模式部署 26](#_Toc356992082)

[3.7.2、 客户端模式部署 27](#_Toc356992083)

[3.7.3、 第三方驱动包方式部署 27](#_Toc356992084)

[3.8、 规划实现模型 28](#_Toc356992085)

[3.8.1、 识别物理组件 28](#_Toc356992086)

[3.8.2、 组织开发工程 28](#_Toc356992087)

[3.9、 架构实现 29](#_Toc356992088)

[3.10、 架构验证和调整 29](#_Toc356992089)

[3.11、 架构编档 29](#_Toc356992090)

[3.11.1、 整理架构目标和策略 29](#_Toc356992091)

[3.11.2、 整理用例视图 30](#_Toc356992092)

[3.11.3、 确定逻辑视图 30](#_Toc356992093)

[3.11.4、 编写架构机制 31](#_Toc356992094)

[3.11.5、 记录架构决策 31](#_Toc356992095)

[四、 尾声 32](#_Toc356992096)

# 说明业务

## 识别业务问题

* 只有清晰的软件结构才能持续地保证软件系统采用合理的成本解决客户新的问题。

* 清晰的软件结构不仅架构师关心，技术管理者也关心。

## 定义软件系统

* 代码中反应出的组件关系将是评价软件结构质量的线索。
* 待建软件系统的主要服务对象是架构师和技术管理者。

# 分析需求

## Cooper要解决的问题

* 解决架构师在跟踪架构实现时发现结构问题不专业、效率慢、有遗漏的问题。
* 解决架构师在对程序结构进行调整时没有明确数据对比的问题。
* 解决技术管理者无法了解到项目实际的程序结构质量问题。

## Cooper需要做到的

* 需要让经验不足的架构师了解到怎样的结构设计才是合理的。
* 需要全面总结不符合结构设计的通用性问题。
* 需要提出一种度量程序结构质量的方法。
* 支持在满足架构师需求的基础上也能够满足技术管理者的需求。
* 支持架构师之间能够交流总结的结构问题和处理办法。

## Cooper的产品特性

* 采用一套遵从业界认可的设计原则而形成的公式来度量程序结构的质量。
* 以图形化的方式直观地展示程序结构中的问题。
* 对发现的结构问题给出调整建议。
* 模拟执行架构师对程序结构进行的调整，并给出数据对比。
* 提供多种（当前25种）分析程序细节信息的分析器。
* 允许用户编写自己的分析器放在Cooper中执行。
* 支持单机模式、客户端模式、浏览器模式、eclipse插件模式运行。
* 服务器端可以导出多个客户端项目的分数列表。
* 服务器端可以记录同一项目的历史数据，并能够向技术管理者提供趋势分析。
* 支持分析器上传和下载。
* 支持分析结果在不同的Cooper间传递。

## Cooper的客户和用户

Cooper有四种应用方式：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方向 | 说明 | 可信度 | 诱惑力（吸金力） | 是否适合作者个性 |
| 面向互联网的Saas服务 | 向各软件公司、定制软件购买组织、开源组织、个人编码爱好者提供有偿代码结构分析服务 | ◆◇◇◇◇ | ◆◆◆◆◆ | ◆◇◇◇◇ |
| 开源项目 | 向开源组织提供该软件及其源代码，识别和组织该领域的编码爱好者共同建设该工具，未来谋求被购买 | ◆◆◆◆◇ | ◆◇◇◇◇ | ◆◆◇◇◇ |
| 公司内推广 | 面向东软，从技术方向推广该工具并建设平台，成为辅助各事业部展开代码检查和指导设计的工具 | ◆◆◆◇◇ | ◆◆◇◇◇ | ◆◆◆◇◇ |
| 独立个人咨询 | 以独立代码结构咨询师的角色使用该工具，为需要该服务的软件公司或组织（如一些东软的政府客户也需要了解、评估购买的代码结构质量）提供代码结构质量咨询服务 | ◆◆◇◇◇ | ◆◆◆◇◇ | ◆◆◆◆◆ |

在公司内推广模式下：

* Cooper卖给技术管理者，架构师和设计师使用Cooper。

## Cooper的功能性需求



* 以导航的方式创建分析组和命令（能够在下次继续使用）。
* 维护待分析的组件模型（能够在下次继续使用）。
* 执行分析命令，查看分析结果（包含质量分数和结构问题）。
* 能够保存分析结果，在下次查看。
* 可以导出分析结果。
* 根据分析结果生成结构调整建议。
* 可以导出调整建议。
* 提供按着调整建议执行虚拟结构重构。
* 提供多种手段（合并组件、拆分组件、删除组件、移动类）执行虚拟结构重构。
* 能够查看结构调整前后的数据差异，并支持在此基础上的再次调整。
* 导出重构步骤和结果。
* 执行分析器查看分析结果。
* 切换工作区【衍生需求】。
* 上传自己编写的分析器和下载他人上传的分析器。
* 向另一个Cooper客户端发送分析结果。
* 与另一个Cooper客户端交流程序结构。
* 系统设置。
* 提供对比多个客户端项目分数的功能。
* 提供导出多个客户端项目的小结报告。
* 提供查看一个项目的历史分数，并可以得到一个趋势预测。
* 监控多个客户端的运行状况。
* 提供多种用于分析多个客户端项目数据的统计分析器。

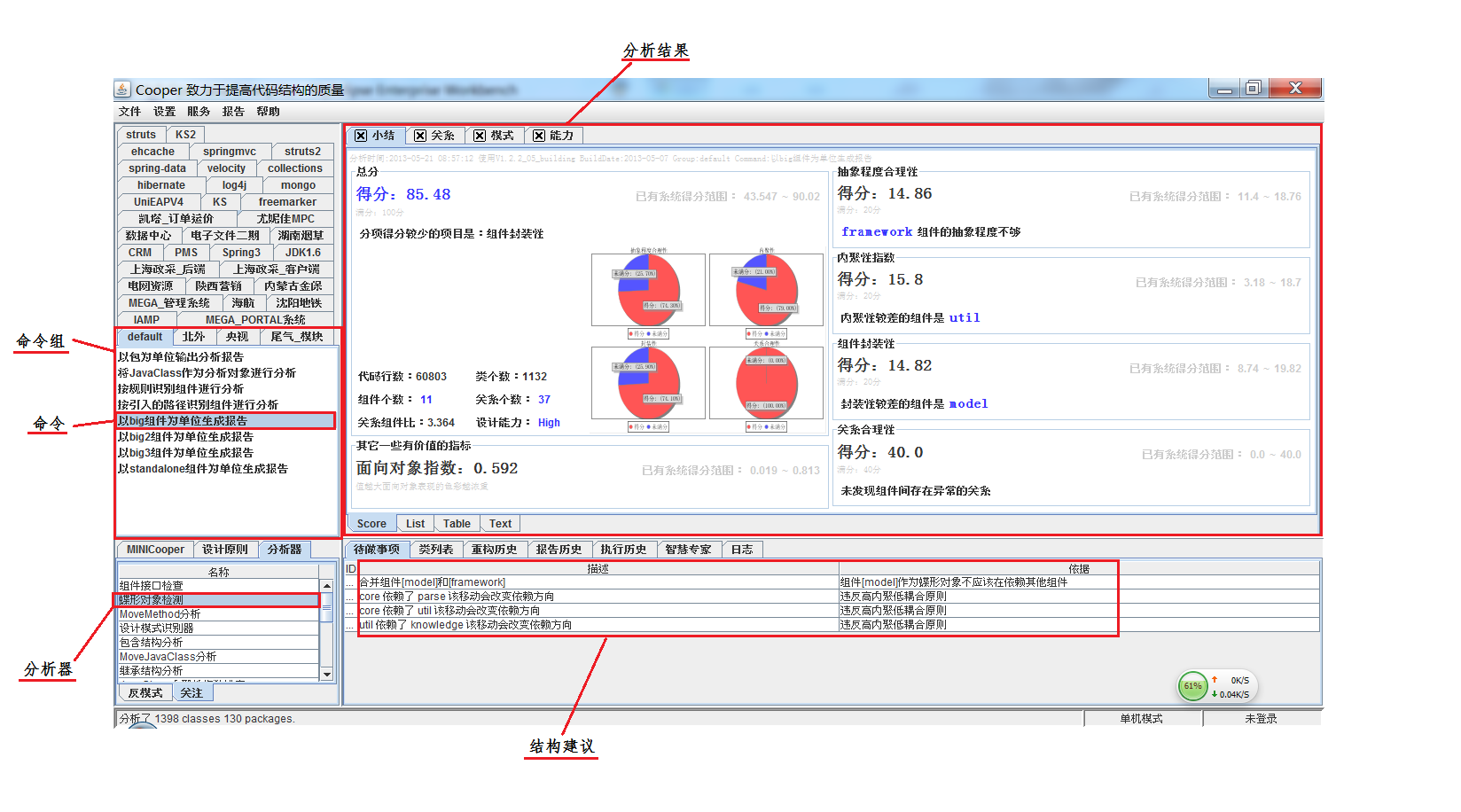
## Cooper的非功能性需求

* 命令执行的时间要求：

2M以下jar在3秒钟以内；2-10M以下jar在30秒以内；10-40M以下jar在60秒以内。

* 分析jar的大小最大需支持到40M。
* 服务端需保证100个在线客户，同时可接收20个客户端执行分析命令。
* 单机版大小需小于8M（能够用邮件发送），无需安装。
* 开发并安装一个简单的分析器的时间在20分钟内。
* 度量模型相关的代码可以以jar的形式集成到第三方应用中。

## 需求中的核心概念



* 分析目标：指待分析的系统代码，可以是多个包含class的文件夹，也可以是多个jar。
* 分析组：为了分析一个分析目标而建立的组，一个分析组对应一个分析目标。
* 组件模型：是将分析目标分成多个分析单元的划分方式，用以得到它们之间的依赖关系，从而判断出这种划分方式下的结构质量。
* 分析命令：对应一种组件模型，可以执行它来查看该划分方式下的结构质量。
* 度量模型：是一种采用分数的形式评价程序结构质量的手段，它循序了业界已认可的设计原则（无环依赖、稳定依赖、稳定抽象等价、高内聚低耦合等）
* 分析结果：包括质量分数，各分析单元的指标值，图形化的依赖关系图，小结报告等
* 虚拟重构：是在内存中模拟的一种结构重构，重构执行后可以查看新结构的结构质量，并能够与之前的数据进行对比。
* 分析器：是一种对分析结果进行个性化分析的方式，它分为关注点和反模式两类，用户可以自己编写分析器对分析结果进行个性化分析。

## Cooper的关键架构需求

* 创建分析组和命令
* 维护待分析的组件模型
* 执行分析命令，查看分析结果
* 根据分析结果生成调整建议.
* 提供按着调整建议虚拟执行结构重构。
* 执行分析器查看结果。
* 支持单机模式和客户端模式运行。
* 单机版大小需小于8M。
* 开发并安装一个简单的分析器的时间在20分钟内。
* 度量模型相关的代码可以以jar的形式集成到第三方应用中。

# 设计架构

## Cooper的架构目标

* 架构必须保证系统能够实现一种由简单到复杂的演变。
* 架构不能引入过多的技术，要严格控制技术复杂度。
* 架构在进行重构时需尽量做到编译时报错而不是运行时出错。
* 架构需保证核心代码高度的复用，当核心代码出错时，可以“到处看到”，反之当在一个场景下没有发现错误时就能够说明对核心代码的修改是没有问题的。

## Cooper的架构策略

* 核心度量模型采用面向对象的风格设计和编写。
* 鼓励使用设计模式。
* 尽量保证在一种技术语言下实现功能。
* 确保核心被人接受后再开展其他功能的编写。

## 确定系统级别的关键交互

* 针对架构师使用的功能有两个关键场景：一个是架构师向Cooper提交需要分析的代码,说明它的逻辑划分，Cooper给出分析结果和调整建议；二是架构师向Cooper提交对结构的调整要求，Cooper给出调整后的分析结果和对比数据。

## 规划系统的高层组织

### 识别分析类

1. 系统级别的交互图中涉及到几个概念：一个是需要分析的代码和它的逻辑组成，我们把它命名为分析目标。除此之外还包括：分析结果、调整建议、结构重构和对比数据，它们之间的关系可以用下图描述：



1. 在关键需求中描述的分析组、命令、组件模型【对应逻辑组成】是对分析目标的细化，主要是在分析目标上增加了配置持久化能力，不需要每次分析都说明分析的代码路径和逻辑组成了。另外，一个分析目标可以有多种逻辑组成。细化了分析目标后的关系如下图所示：



* 在关键需求中”创建分析组和命令”和” 执行分析命令，查看分析结果”中的两个命令含义是不同的。前者是创建一个命令配置，并且需要持久化，后者是按着配置命令执行一次分析动作。
* 执行一次分析动作需要大量的计算，并且该计算在客户端模式下需要在服务端执行，于是后面需要有一个分析服务，负责产生分析结果。该部分应该设计成一种分层的结构风格。
* 创建分析组和命令、维护组件模型、查看分析结果需要有一个界面操作和展示，暂时可以把它们统一建模成一个边界类。
* 分析结果的展示涉及到表格、文本、图形等多种方式，还包括输出格式的差异（swing和html），并且独立于数据本身，可以参考MVC模式识别出一个独立的报告生成器来负责分析结果内容的展示。
* 需要有个控制器负责结构建议的产生。虚拟执行器主要负责建议的执行。

1. 这样关键概念（实体类）将进一步演进，增加边界类和控制类，如下图所示：



* 增加了UI界面边界类统一指代用户操作的界面。
* 新增两个控制器负责配置信息的维护（命令配置控制器）和命令的执行（命令执行控制器）。
* 增加了分析服务控制器，负责在服务端分析数据产生分析结果。
* 报告生成器、结构建议生成器、虚拟执行器和分析器都是基于分析结果的。

### 初步划分分析包

1. 为了保证客户端模式的运行，需要规划那些控制器在客户端运行，那些控制器在服务端运行。参考生产者和消费者模式，分析结果正是生产者和消费者共用的数据。于是将分析服务划分到服务端运行，其他划分到客户端运行。如下图所示：



1. 服务器除了提供分析服务外，也需要提供UI监控客户端的运行，导出多个客户端的分析结果，于是给服务器增加边界类。该边界类不仅可以在窗口模式下运行，也应该支持命令行模式。度量程序结构的质量需要一套公式进行计算，该计算需要了解分析结果中的数据，于是由分析结果来负责度量计算。下面是增加了这些分析类后的分析类图：



* 基于度量模型，建模了分析单元间的依赖关系，该依赖关系对度量结构质量，发现结构问题有非常重要的贡献。
* 建模了分析单元下的类，因为分析单元间的关系实际上就是分析单元中类之间的关系，这些关系有类型之分，表现出不同的强度。

### 考虑分析机制

* 分析组和命令配置信息、组件模型、分析结果需要【持久化】。

配置信息是一种简单、小量的有层次格式的信息；分析结果是一种复杂、大量的信息，并有检索需要。两种数据的持久化机制应该有差异。

* 客户端和服务器交互需要【进程间通讯】。

考虑非功能性架构需求“支持单机模式和客户端模式运行”和“单机版大小需小于8M”，该进程间通讯机制应尽量简单，并保证运行模式相关的代码对于其他模块是透明的。

* 报告生成器需要【图形展示底层服务】。

该图形展示需要支持表格、饼状图、柱状图等展现手段，并且能够展示分析单元间的关系，标识出有问题的关系等。

* 命令执行控制器需要【异步】执行，并能够监控进度。
* 还有一些常规的【配置】、【日志】、【错误处理】等机制。

### 精化分析包

1. 补充分析包。将分析机制相关的通用内容组织成基础设施包。在结构建议生成器、虚拟执行器和分析器中会用到未来具有商业价值的部分，即分析结果数据库和知识专家。增加给开发者使用的用于统计分析分析结果的包。增加了这三部分后的分析包结构如下图所示：



* 将知识库中的对外服务识别为领域专家，内部采用多个领域分析器来组织领域逻辑。
* 将分析结果存取器由服务器移入到知识库中，让知识库来负责分析结果的存取。
* 统计分析提供界面，将不同的分析项目建模为统计分析器。
* 统计分析器依赖异步、图形服务和分析器。

1. 分解分析包。目前比较大的分析包是客户端，它需要进一步的分解。考虑复用，首先将结构建议生成器、虚拟执行器和分析器独立成一个包， UI界面、命令配置和执行形成一个包，报告生成器形成一个包。为了简化分析器的安装，工具可以通过动态加载分析器的方式安装【动态加载】，这样客户端需要不通过分析服务就可以独立使用到解析器，由此需要将解析器独立成一个分析包。分析后的分析包图如下所示：



* 将客户端一分为三：命令配置与执行、报告生成和通用领域工具。
* 将服务器一分为二：服务器和解析服务。
* 新增报告历史存取器，负责文本报告的存取。
* 新增分析器管理器，用于动态加载分析器，它使用到了解析服务。
* 补充分析机制：动态加载。
* 新增类构建器，负责创建类和类关系。

### 描述用例实现

1. 描述执行分析命令。

下图是执行分析命令的主流程：



* 命令执行控制器为主控制器，协调命令的执行、报告的产生和结构建议的产生。
* 命令的创建需要调用命令配置控制器得到的命令配置信息创建。
* 报告生成器创建报告后直接显示到UI界面上，之后再生成结构建议信息。

下图是命令执行的子流程：



* 分析服务是总控制器，用于协调解析器、分析单元组织器、分析结果存取器，创建和保存分析结果。
* 解析器负责创建类及其关系。
* 分析单元组织器负责创建分析单元和建立与类之间的关系。
* 分析结果存取器负责持久化分析结果。
* 分析结果在创建时建立分析单元之间的关系。

下面是报告生的子流程：



* 报告涉及到多个部分，不同的部分涉及到多种格式。将部分建模为控制器，将方式建模为控制器上的方法。

下面是生成结构建议子流程：



* 结构建议生成器需要调用领域专家获取结构建议，通过识别结构问题创建建议条目，并能够按着重要程度排序。

## 演进系统的高层组织并形成组件模型

### 初始化组件模型

1. 分析已经形成的分析包的职责内聚性，规划组件接口，形成初始的组件模型。



* 将UI界面边界类建模为ui包。
* 将命令配置和组件模型建模为config组件，提供命令和组件模型的配置接口。
* 将命令执行建模为command组件，负责协调其他组件执行命令。
* 将分析服务建模为service组件。
* 将服务器建模为serverui包，通过接口ServiceMonitor监控服务执行。
* 将解析器服务建模为parse组件。
* 将知识库建模为knowledge组件，提供Expert接口负责结构建议服务，AnalysisResultRepository接口负责存取分析结果。
* 将报告生成建模为report组件。
* 将通用领域工具建模为util组件，提供TODOListIdentify负责生成结构建议，RefactorTool负责执行重构，AnalyzerExecutor负责执行分析器。

1. 为了降低模式切换对（单机模式和客户端模式）其他组件的影响，在command组件后设计了一个serviceproxy组件来隔离通讯方式的变化。为了使其能够支持浏览器模式访问，需要引入新的包来接收http请求。这样组件模型进一步演进为下图所示：



* 增加了serviceproxy组件，负责隐藏调用模式的变化。
* 增加了web包，负责接收http请求（目前未考虑复用command）。

### 确定设计机制

* 持久化
* 对于命令配置信息这样有层次格式的小量数据采用xml格式进行存储，在统一的管理器（ConfMgr）中实现对私有配置内容的存取。管理器设计成一个单例，在管理器启动时装载配置信息，在内容发生变更时持久化配置信息，同时通过调用ConfChangeListener列表中的监听器向注册者发出通知。调用序列如下图所示：



* 对于分析结果摘要这样大量且有检索需求的信息采用关系型数据库的形式进行存取。通过Repository来存取这种信息。
* 对于分析结果这样复杂、大尺寸且大量的数据，将采用序列化并结合压缩的手段进行存取（目前的存储手段是关系型数据库，未来考虑采用mongoDB）。在反序列化的过程中需要实现其回调函数对计算数据进行初始化。其包含的子对象都需要采用该种方式进行计算数据的初始化。
* 对于分析器、启动配置信息这样的小量、非结构化数据，将采用序列化手段以文件的形式进行存取。
* 进程间通讯

客户端与服务器间通讯将设计成请求响应型的交互方式，并且在服务器端通过Session保留客户端的会话信息。以下是交互图：



* 客户端采用RemoteSessionProxy负责与远程SessionService交互。
* 服务器端通过SessionService、SessionMgr和SessionListener完成会话管理。
* SessionMgr当启动时创建守候线程负责会话过期的逻辑处理。
* 每次请求前客户端通过RemoteSessionProxy判断会话是否有效，如果无效，自动匿名登录。
* 每次请求都需要构建Request对象作为远程服务的参数。
* 图形展示底层服务

在framework中创建一个统一的工具类负责提供图形展示功能。传入的是图形数据（包括数据和格式信息），返回的是已组织好的图形对象。

* 异步执行

异步执行将设计一个抽象的Action来规范需要异步执行的动作，该抽象类包含进度条监控功能。子类可以通过继承该抽象类，实现其方法来执行异步任务。

* 错误处理

错误处理主要以抛异常的方式实现，系统中有一个统一的异常基类，不同组件的异常可以通过继承该基类声明自己的异常类型。

* 动态加载

动态加载指的是对于同一类型的子类,只要在启动时放在classpath中,无需注册就可以使用到它。通过一个管理器负责可以读取到该类型的所有子类的集合。目前的分析器采用该种方式进行无注册下直接使用。

### 精化组件模型

1. 确定完设计机制后，组件模型进一步得到演进：



* 增加用于在客户端模式运行时需要的sessionproxy和session组件。
* 向framework中增加了提供服务的工具类、技术组件接口和有规范性质的抽象类。
* serviceproxy需要通过和sessionproxy的协调完成进程间通讯。

1. 细化model包。将分析模型中的实体类进一步细化为设计类。组件模型进一步演进到下图：



* 通过领域分析抽象了Scored（可被分数描述的）、Measurable（可被指标度量的）两个接口。
* 抽象类AnalysisResultScored和AbstractJDependUnit负责实际分数和指标值的计算。
* 抽象类Component对应分析单元，负责将JavaClass组织成分析单元，其子类需要实现组织分析单元的逻辑。
* 分析单元间的关系被建模为Relation，负责计算关系的强度、类型、受关注的程度，以及判断是否存在问题。
* 类之间的关系建模为JavaClassRelation，其子类描述了5种类关系。
* 为了更深入的支持分析器逻辑的编写，识别了Attribute、Method和InvokeItem三个类，对属性是否存在状态、方法覆盖、调用序列提供了丰富的API支持。

### 描述用例实现

1. 描述执行分析命令。

下图是执行分析命令的主流程：



* UI界面分析类在该场景下被建模为CommandPanel。
* 设计了CommandAdapter通过协调CommandConf和Command完成产生分析结果的职责。
* 当创建CommandPanel时通过CommandConfMgr加载了全部的CommandAdapter。
* TODOListIdentify. Identify在CommandAction的异步线程中执行。

下图是命令执行的子流程：



* 在CommandAdapter通过创建Command调用ServiceProxy。
* AnalyseRemoteService负责处理通讯协议，实际的分析服务仍然委托给AnalyseLocalService。
* Parse负责完成类及其关系的构建。
* Component负责完成分析单元的构建。
* AnalysisResult负责分析单元关系的构建。

## 考虑系统的运行时模型

### 单机模式

在单机模式运行下，所有的组件都会运行在一个进程中，包括客户端、服务器、知识库。存储分析结果摘要的数据库服务也需要在该进程中运行。下面是一个示意图：



* 多个单机模式运行的Cooper可以通过Socket传递分析结果，进行文字交流。
* 单机模式下不包含sessionproxy、session等用于通讯的组件。

### 客户端模式

客户端模式运行时，客户端是一个胖客户端，服务器是一个独立进程。客户端分数存取需要的数据库服务需要运行在客户端进程中，服务器中需要的数据库服务需要独立运行。知识库作为一个分析型服务也需要独立运行。下面是一个示意图：



* 客户端进程包括ui、config、parse、command、commandproxy、util、report、model和framework组件和包。
* 客户端通过rmi与服务器通讯。
* 服务器包括serviceui、service、parse、session、model和framework组件和包
* 服务器是一个独立的进程，需要支持有界面运行和命令行运行。
* 知识库作为独立的进程运行，有自己的管理界面。
* 服务器端的数据库服务独立运行。

### 浏览器模式

在浏览器模式运行时，服务器端运行在web容器中，通过web包接收客户端请求。除knowledge和statistics外，其他组件全部运行在web容器中。下面是一个示意图：



* WebServer包含了web、serviceproxy、service、parse、model和framework组件和类。
* 客户端会包含applet或flash等性质的组件。

### 插件模式

Cooper可以被集成到eclipse中，以插件的形式运行。该种模式运行时，本身没有独立的进程，包含的组件有serviceproxy、service、parse、util、knowledge、model和framework组件。另外需要为插件模式开发前端组件。

## 规划系统的部署模型

### 单机模式部署

当系统以单机模式部署时，整个系统只有一个exe文件，不需要安装。当第一次启动时，系统要求选择工作区，在选择的工作区中创建系统需要的配置信息、数据库文件、日志文件，并将选择的工作区信息以一个文件的形式放在exe的目录下。当系统启动后，用户对系统的操作产生的业务信息将都存储在该工作区中。工作区中的内容可以在另一个Cooper单机模式运行的系统中使用。



* 该部署方式要求运行的机器安装有JRE。

### 客户端模式部署

客户端模式部署时，客户端仍然需要部署一个exe，服务器端需要三台机器，一个机器部署分析服务exe和知识库exe，一个机器部署数据库服务，一个机器部署统计分析exe。



* 当客户端启动后需要配置远程服务器的地址。
* 客户端需要登录服务器成功后才可以执行命令。
* 每个客户端的配置信息都会存储在各自机器上的工作区中。

### 第三方驱动包方式部署

第三方可以将Cooper作为底层驱动包的形式集成到自己的服务中。在免费模式下包含的组件有service、parse、model和framework，付费模式新增util和knowledge，其中knowledge将与远程知识库服务通讯已获得基于大量数据的知识专家的结构建议。

## 规划实现模型

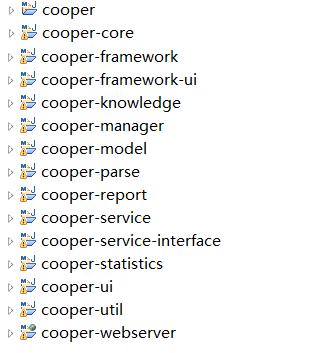
### 识别物理组件

在规划系统的部署模型时，不同的部署方式需要的组件有较大的差异，为了更灵活的满足在不同部署模式下只包含必要的代码，需要对目前已经形成的组件模型，按着部署模式的要求进行调整。

* 将framework包中包含的ui相关的代码分离出来，形成一个framework-ui物理组件，以避免在第三方驱动包模式部署时不部署该部分代码。
* 将service逻辑组件分离为service-interface和service两部分，在客户端仅部署service-interface即可。保证service-中的代码的安全性（含防作弊器子组件avertcheat）。

### 组织开发工程

目前已识别出的组件个数达到了20个以上，不同的部署模式需要发布的组件差异很大，需要采用一种比较高效的方式来管理这些组件。maven是一个不错的选择。系统在开发期将以组件的粒度被换分成多个工程。下面是被划分成的开发工程：



* core工程包含了config、sessionproxy，以及客户端需要的其他基础组件和包。
* manager工程对应serverui包。
* service工程包含了service、session、avertcheat，以及服务器需要的其他基础组件和包。
* webserver工程对应web包。

每一个工程将打包成独立的jar，在不同的部署方式下选择不同的jar进行部署。

## 架构实现

1. 按着开发工程的规划创建实际的maven project。
2. 将设计模型中的设计类导出成代码，并发布到所属工程下的src中。
3. 按着用例实现中描述的调用流程补充框架性代码。
4. 按着需要引入第三方jar。
5. 在编写框架性代码时验证架构设计是否合理。

* model包中的CustomComponent将直接使用到config下的ComponentModelConf,这样就形成了model包对config组件的依赖，为了解决该依赖暂时将组件配置相关的代码移入到了model中。

1. 运行单元测试，验证调用序列。

## 架构验证和调整

1. 采用不同大小的jar和class文件路径在开发出的框架系统中测试，识别性能热点，调整设计。

* class文件扫描是一个性能热点，通过改进FileGatherUtil中的算法（并发）提高性能。
* JavaClassBuilder中类关系的建立JavaClassRelationCreator是一个性能热点，通过分离处理阶段、改进数据结构提高性能。
* 当计算分析单元指标时，存在重复计算，通过引入计算列缓存计算结果来提高性能。
* 将分析结果和分数的保存设计成异步来提高主流程的性能。
* 考虑改进report组件一次性产生全部报告的方式来提高用户感受。

1. 当系统发生错误时，通过改进异常处理的机制保证系统的健壮性。

* 在AbstractParser.parse中通过捕获顶级异常避免一个class文件解析的错误导致分析结果无法产生。
* 在产生分析报告时对于局部的不同类型的报告块也通过捕获顶级异常避免因一个子区域的错误无法看到分析报告。

## 架构编档

### 整理架构目标和策略

* 适度提前浏览器模式或插件模式的开发，以提高大家对产品的接受速度。

### 整理用例视图

向用例视图中增加技术管理者和管理员参与者，并识别他们使用的用例。



### 确定逻辑视图

将最终的组件模型作为逻辑视图。



### 编写架构机制

整理设计机制将其编写到架构文档中的架构机制部分。

### 记录架构决策

整理架构决策将其编写在架构文档中。

* 系统不使用spring、hibernate等主流开源框架，以自行编写和jre有的功能为主。
  + 理由：保证Cooper尽量小，开源项目很多特性使用不到。
* 系统采用prefuse开源项目作为关系信息的图形展现底层驱动。
  + 理由：数据驱动、开放，有丰富的layout。
* 客户端模式中客户端和服务器采用rmi作为通信协议。
  + 理由：保证Cooper尽量小，编码风格比较一致。
* 数据库采用hsqldb
  + 理由：支持In-Process模式和服务器模式。

# 尾声

* 架构设计的过程是一个分解的过程，本文叙述的分解的思路有别于仅仅考虑按功能进行分解的思路，这种差别主要表现在两个方面：
* 分解的过程将功能性需求和非功能性需求一并考虑，它们都是架构需求。
* 分解的线索不仅来源于用例，更重要的是来源于领域核心概念，通过分析核心概念的内涵、关系的强度、稳定性差异等因素来进行基于核心概念的分解。

由于以上两个方面的差异，使得被分解后的多个部分之间有紧密的协作，由于各自的职责不同，设计的风格和代码的形态上差异较大，这样造就了一个有机的、丰富的、完整的软件结构。这种分解避免了烟囱竖井式的系统风格，也使得软件架构师的工作真正可以有效实施，深入影响软件的结构，并最终可以为系统的非功能性需求负责。