

分类号: R319

单位代码: 10335

密 级: 无

学 号: _____

浙江大学

硕士专业学位论文



中文论文题目: 可动态扩展的医疗数据集成可视化系统
设计与开发

英文论文题目: The Design and Development on
Extensible Integrated Visualization
System of Medical Data

申请人姓名: _____

指导教师: _____

专业学位类别: 工程硕士

专业学位领域: 生物医学工程

所在学院: 生物医学工程与仪器科学学院

论文提交日期 2016 年 01 月 05 日

可动态扩展的医疗数据集成可视化系统

设计与开发



论文作者签名: _____

指导教师签名: _____

论文评阅人 1: _____

评阅人 2: _____

评阅人 3: _____

评阅人 4: _____

评阅人 5: _____

答辩委员会主席: _____

委员 1: _____

委员 2: _____

委员 3: _____

委员 4: _____

委员 5: _____

答辩日期: _____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 浙江大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

签字日期：

年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：

导师签名：

签字日期： 年 月 日

签字日期： 年 月 日

摘要

医疗数据集成可视化系统是对医疗数据进行集成浏览的信息系统，目的是在一个屏幕下通过一定的导航或索引方式对临床医疗数据进行集成浏览。随着医学技术的快速发展，临床上医疗数据的种类快速增多；另外，由于临床业务日益复杂，不同的临床场景对数据的集成浏览需求不同。这些都要求医疗数据集成可视化系统能动态扩展，以满足医疗数据种类不断增多以及数据集成浏览方式随临床场景变化的需求。

国内外在医疗数据可视化系统动态扩展方法研究方面已经有了一定的研究成果，但是在实际应用中仍存在如下两个问题：系统与底层数据的耦合程度太高，系统扩展的同时往往伴随着大量数据库接口的开发；系统各扩展模块之间缺少灵活的通信支持，交互能力不足。

针对以上问题，本论文对可动态扩展的医疗数据集成可视化系统展开研究，主要内容如下：

- (1) 针对系统与底层数据库高耦合的问题，本论文基于 openEHR 两层模型方法实现了一种数据绑定机制，通过 openEHR 原型查询语言 AQL 实现数据获取，避免了频繁的底层数据库接口的开发。
- (2) 针对扩展模块之间交互能力不足的问题，本论文设计了一种插件式动态扩展框架，并通过提供各模块间灵活通信接口的方法实现各扩展模块之间的交互。
- (3) 基于上述研究，本论文设计并开发了一套医疗数据可视化系统，同时开展了临床应用开发实践。实践结果表明，系统能很好的实现动态扩展要求，各扩展模块之间也可以实现灵活的交互。该系统已在国内某三甲医院所有门诊上线应用，很好地满足了门诊医生不同临床场景下的需求。

关键词：医疗数据 集成可视化 数据绑定 动态扩展 openEHR 插件

Abstract

The Integrated Visualization System of Medical Data is an information system to browse the clinical data in a certain navigate mode within a screen. With the rapid development of health care technologies, the types of the clinical data are grouping rapidly and with the increasingly complex of the clinical business, different clinical scenes have different needs of the data integrated view. Taking these observations into consideration, the Integrated Visualization System of Medical Data needs to have extensibility.

Although the extensible of the Visualization System of Medical Data has been some studies, but the following problems are still included in the practical application: the coupling of the system and the underlying database is too high, system expansion is accompanied by a large number of database interface development; the system lack of good interactivity between the extension modules.

To the problems mentioned above, this thesis does a research on extensible integrated visualization system of medical data which includes:

- 1) To the problem of high coupling between the system and the underlying database, this thesis has proposed a method of data binding based on openEHR two-level model approach. The system get data by openEHR Archetype Query Language (AQL). This method can avoid the frequent development of database interface.
- 2) To the problem of lack of interactivity between the extension modules, a plug-in dynamic extension framework for the system is designed. The extension modules interact with each other by a flexible communicate interface.
- 3) Based on above research, this thesis designed and developed a visualization system of medical data and the system was used in clinic. The practice shows that the system can address the requirement of extensibility and expandability and the extension modules can achieve flexible interaction with each other. The system has been used in the outpatient department of a domestic top three hospital, it can meet the needs of different clinical scenarios in the outpatient department.

Key Words: medical data; integrated visualization; data binding; extensible; openEHR; plug-in

目录

摘要	I
Abstract.....	II
目录	III
1.绪论	1
1.1 课题背景	1
1.2 医疗数据集成可视化综述	2
1.3 研究目标与内容	6
1.4 本章小结	7
2.数据绑定机制设计	1
2.1 数据绑定需求分析	1
2.2 技术背景介绍	1
2.2.1 JSON 知识介绍	1
2.2.2 openEHR 规范介绍	6
2.3 数据绑定机制	7
2.3.1 数据绑定原理	7
2.3.2 基于 openEHR 原型的数据绑定方法	8
2.4 本章小结	11
3.系统动态扩展框架设计	12
3.1 系统总体结构	12
3.2 动态扩展技术研究	13
3.2.1 COM 组件技术	13
3.2.2 .NET 插件技术	14
3.2.3 扩展技术选择	17
3.3 插件式动态扩展框架设计	18
3.3.1 基于插件的系统扩展框架结构	18
3.3.2 系统成员分析	20
3.3.3 插件间通信方法	21

3.3.4 插件接口设计	22
3.4 本章小结	25
4.系统实现与临床实践	26
4.1 系统模块设计与开发	26
4.1.1 插件发布模块	26
4.1.2 插件更新模块	30
4.1.3 插件引擎模块	31
4.1.4 系统监控模块	32
4.2 临床应用实践	34
4.2.1 可视化组件开发	34
4.2.2 集成视图插件开发	37
4.2.3 视图个性化订制	42
4.3 本章小结	43
5.总结与展望	44
5.1 总结	44
5.2 展望	45
参考文献	46
作者简介	49
附录一	50

1. 绪论

1.1 课题背景

医疗数据集成可视化系统，是对医疗数据进行集成浏览的信息系统，它由不同的集成视图组成。集成视图，是指为满足在一个屏幕下展示医生所需要的医疗数据而通过一定的导航或索引方式对这些数据进行集中展现的可视化视图。

集成视图的设计准则是能做到让医生在同一屏幕下获得病人病情、诊疗的概况，就是做到将尽量多的有效医疗数据展现到屏幕上，有效医疗数据是指医生在某个医疗场景下所关注的医疗数据。例如，会诊场景下，病人的既往检查、检验数据是医生所关注的的数据，这部分数据即为有效数据，应该在屏幕上尽可能多地展现这部分数据。现存的医疗数据的集成可视化往往将所有类别的数据统统显示在屏幕上，没有考虑不同医疗场景下医生所关注医疗数据类别的差异性，同时，医疗数据的扩增将会使有效数据的查找更困难，这便是信息过载的问题。

另外，随着医院信息化的发展，医疗数据变得越来越庞大^[1]，主要表现在两方面：

1) 医疗数据的种类越来越多

医院已有的医疗数据大致如图 1.1 所示，不过这还不是临床数据的全集，医院的临床信息系统的建设是一个循序渐进的过程，临床信息系统建设的过程就是医疗数据种类不断增长的过程，随着医院的发展不断完善，一些专科化、个性化的数据也会逐步地补充到电子病历数据集中。例如，生物治疗技术的发展促使大量的生物治疗数据补充到电子病历数据集中，个性化医疗的发展促使大量的基因组数据补充到电子病历集中。另外，随着互联网（尤其是移动互联网）的发展，产生大量日常生活健康数据，医疗领域正逐步进入大数据时代。这些数据在大大丰富医疗健康数据的同时，也将给医疗数据的可视化带来更大挑战。



图 1.1 临床医疗数据

2) 医疗数据的量越来越大

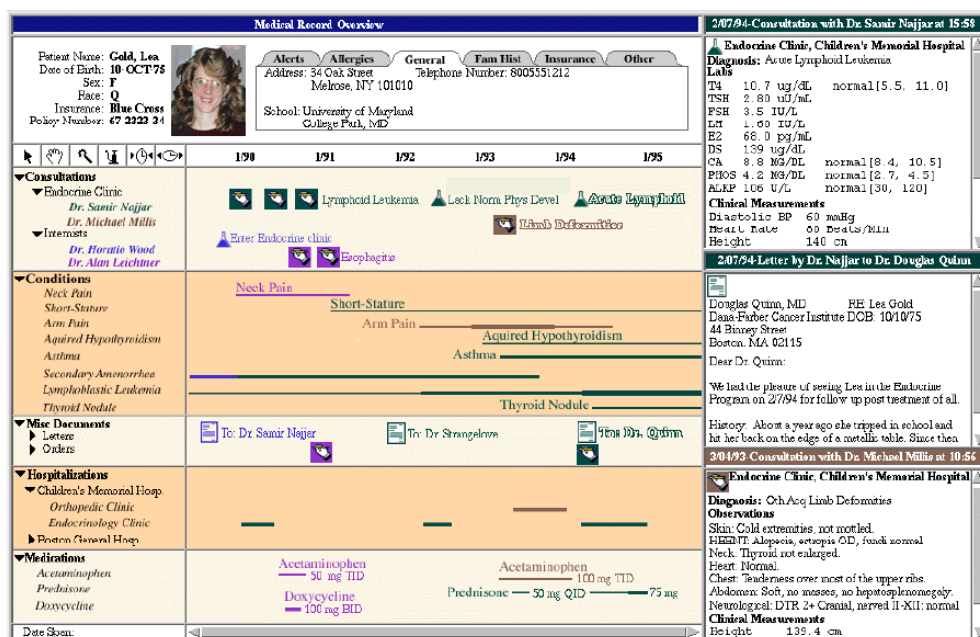
随着时间的增长，病人数据会逐渐增多，对于单个病人来说，个人就医记录的增多将会导致其医疗数据在时间维度上变得越来越庞大。

如何去适应这种医疗数据快速扩增的情况，成为了现阶段医疗数据集成可视化系统进一步研究和发展的重点。

1.2 医疗数据集成可视化综述

医疗数据作为医疗诊断的重要依据，对医疗业务具有极其重要的作用，病人的历史医疗数据对医生的诊断治疗具有重要的参考价值^[2]。在临床数据种类繁多，数据量庞大的情况下，医疗数据的集成可视化成为了研究的热点。

国外对于医疗数据集成可视化的研究比较早，如在上世纪 80 年代由 Tufte 和 Powsner 提出的 Time Lines 的方法^[3]。90 年代由 Catherine Plaisant 等人提出的 LifeLines 的方法，它是对 Time Lines 方法的延伸^[4]，如图 1.1 所示，纵向以医疗事件分组，横向为时间轴，病人的数据根据时间的顺序用缩略图形的形式显示在对应的分组中，通过这个视图可以在一个屏幕下浏览该病人的大致就诊情况，医生可以通过点击缩略图标来显示数据的详细信息。

图 1.1 LifeLine 可视化视图^[4]

2002 年 Bui 等人最早明确提出了“集成可视化”这一术语，他们结合 TimeLine 方法，针对泌尿科病人开发了针对医疗文档和医学影像等数据的集成可视化视图^[5]。之后，Bui 等人在之前的研究基础上又将病历数据按层次进行了组织，基于 TimeLine 的方式对数据按层次进行集中展现^[6,7]。

IBM 利用三维可视化技术通过三维人体器官模型对电子病历数据进行导航^[8,9]。近期，Modernizing Medicine 开发了实现于移动端的电子医疗助理（electronic medical assistant, EMA）产品^[10]，该产品主要服务于中小诊所，提供了交互式的可自由缩放的三维层状解剖图，如图 1.2 所示，其中含有 36000 个热点，医生可以通过触屏的形式掀开皮肤露出肌肉、关节和肌腱，并通过轻触热点显示对应部位的医疗数据，值得一提的是，该系统允许医生在热点所对应部位标注信息并将其数据存储在云端。

图 1.2 电子医疗助理（EMA）^[10]

同时, Google 实验室研制成功了 Google 人体浏览器 Zygote Body^[11], 如图 1.3, 就像 Google 地图一样, 人们可层层深入人体, 从皮肤到肌肉, 再到骨骼(器官), 最后到心血管和神经系统等各个部位, 2012 年 Google 宣布该项目开源^[11], 人们可以在网络上获得该项目的全部源代码。Google 人体浏览器对人体各解剖学部位实现了完美的展示, 在业界引起关注, 同样对电子病历数据的三维人体集成可视化的研究也有很大的激励作用。

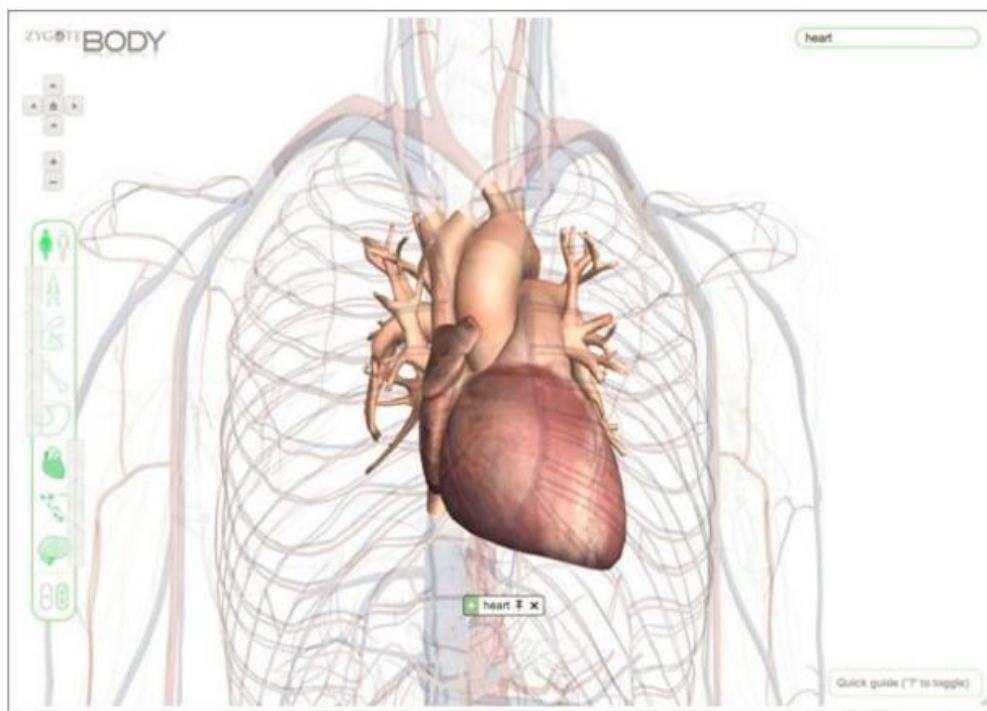
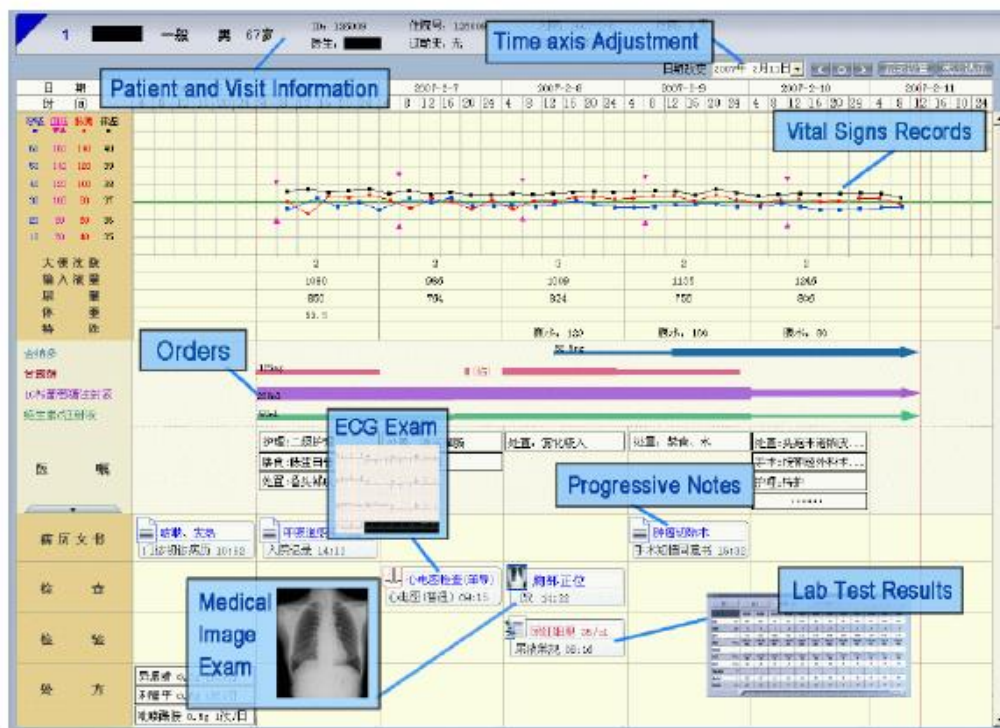


图 1.3 Google 人体浏览器^[11]

近些年, 随着我国医疗信息化的快速发展, 国内也开始展开对医疗数据集成可视化的研究, 主要包括“时间-事件”的导航方式^[1], 如图 1.4 所示, 即以横向时间轴, 纵向医疗事件的方式组织数据, 目的是能做到让医生在同一屏幕下获得病人病情、诊疗的概况。

图 1.4 “时间-事件”的集成可视化^[1]

随着以电子病历为核心的临床一体化格局的形成, 医疗数据的集成可视化也越来越多地体现在电子病历系统的设计当中, 图 1.5 为电子病历系统中就诊导航视图, 该视图左侧部分列出了病人所有的就诊记录节点以及对应就诊记录中包含的所有的医疗数据节点, 选择对应节点可以在右侧区域显示详细信息。需要说明的是, 该视图是电子病历系统医生端的业务操作 (如下医嘱、开检查、开手术等) 视图, 虽然不是专门为数据的集成浏览而设计的, 但无处不体现着集成可视化的思想。



图 1.5 电子病历系统就诊导航视图

1.3 研究目标与内容

医疗数据的快速扩增要求医疗数据集成可视化系统具有足够的灵活性和扩展性。首先,需要能实现新增数据类型的动态可视化扩展;另外,需要满足不同临床场景的临床数据集成视图的动态扩展。

对于医疗数据集成可视化系统的动态扩展方法,国内已经有一些研究,如陈湖山提出的一种集成视图的动态配置方法^[7],通过可视化组件的形式来实现新增医疗数据的可视化扩展,同时还提供了一种集成视图模板编辑器,如图 2.5 所示,在该编辑器中可以对可视化组件进行拖拽来组装自己希望的集成视图,在模板编辑器中可以对组件编写脚本,实现视图的交互特性。这种方法具有一定的灵活性,能在一定程度上实现快速设计并扩展集成视图的功能。但也存在一些问题:

- 系统与底层数据的耦合程度太高,系统扩展的同时往往伴随着大量数据库接口的开发。
- 系统各扩展模块之间缺少灵活的通信支持,系统交互能力不足。

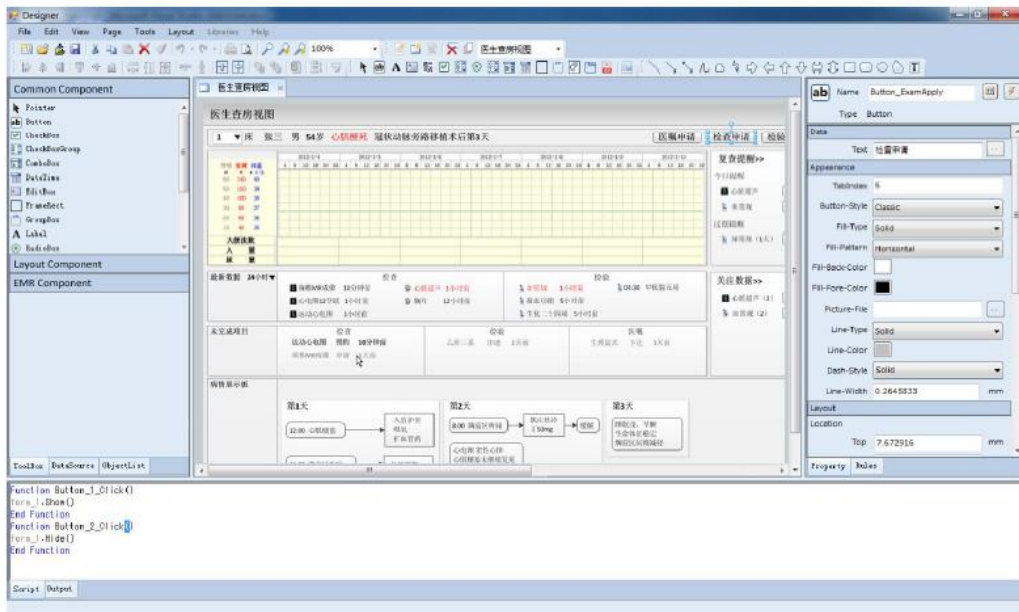


图 2.5 集成视图模板编辑器^[7]

针对系统与底层数据库高耦合的问题,本论文设计了一种基于 openEHR 两层模型的数据绑定机制实现系统与数据库底层的低耦合。针对扩展模块之间交互能力不足的问题,本论文设计了一种基于插件的动态扩展框架,系统的扩展包括集成视图的扩展和可视化组件的扩展,通过为插件开放通用通信接口的方式来实现各扩展模块的相互通信,增强系统的交互特性。

在系统框架设计的基础上，对系统功能进行了设计和开发，并在国内某三甲医院门诊进行了临床应用实践。

1.4 本章小结

本章首先对医疗数据集成可视化系统的定义进行了说明，并对本论文的研究背景进行了介绍，说明了医疗数据集成可视化系统的动态扩展需求。然后对国内外医疗数据集成可视化的研究现状进行了介绍，最后通过分析确定本论文的研究目标：

- 通过数据绑定机制实现医疗数据与底层数据库的低耦合。
- 设计一种动态扩展框架，实现各扩展模块之间的灵活交互。

2. 数据绑定机制设计

2.1 数据绑定需求分析

在传统软件开发模式下，如果有新的医疗数据产生，开发人员首先需要根据临床业务去组织数据的底层存储，然后在软件开发过程中，开发人员需要对业务进行抽象并开发数据库访问接口来支撑上层业务，如图 2.1 所示，这种上层应用与底层数据库的高耦合大大降低了系统扩展的灵活性，尤其是在那些底层数据库对软件开发人员不公开的系统中，需要团队合作才能实现动态扩展。

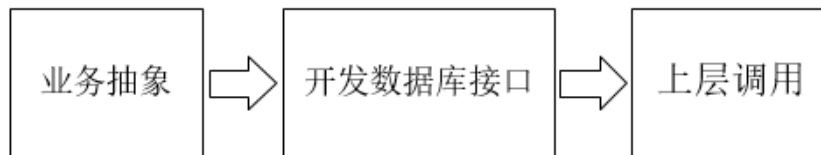


图 2.1 数据库接口开发

为了降低系统与数据库底层的耦合，本章设计了一种数据绑定机制，通过数据绑定，可以避免图 2.1 所示的工作并实现数据的访问，该数据绑定机制是基于 openEHR 两层模型实现的。

openEHR（open Electronic Health Record，开放电子健康档案）是开放的医疗卫生信息标准规范，它提出的目的是让领域专家参与到软件的开发过程中，实现领域知识驱动软件开发，openEHR 标准提供了一种两层建模方法，可实现领域专家对临床业务的建模和抽象，同时 openEHR 官方还提供了一种基于原型的数据查询语言（AQL）^[12]，实现基于原型的软件开发，下一节将对 openEHR 规范进行详细介绍。

2.2 技术背景介绍

2.2.1 JSON 知识介绍

JSON 是一种轻量级的数据交换语言，易于阅读和编写，同时也易于机器解析和生成^[13]，它由道格拉斯·克洛克福特所设计。JSON 采用完全独立的文本格式，但是也使用了类似于 C 语言家族的习惯。这些特性使 JSON 成为理想的数据交换语言^[14, 15]。

1) JSON 的结构

JSON 建构于两种结构^[14]：

- “名称/值”对的集合。不同的语言中，它被理解为对象（object），记录（record），

结构 (struct)，字典 (dictionary)，哈希表 (hash table)，有键列表 (keyed list)，或者关联数据组 (associative array)。

➤ 值的有序列表。在大部分语言中，它被理解为数组 (array)。

这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

JSON 具有以下这些形式：

- a) 对象是一个无序的“‘名称/值’对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’对”之间使用“,”（逗号）分隔。

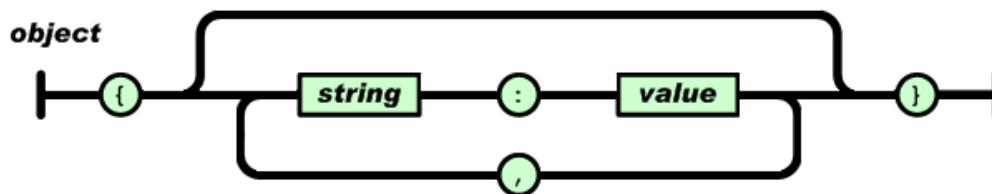


图 2.2 JSON 对象^[14]

如下，表示人的一个对象：

```
{
  "name": "张三",
  "age": 24
}
```

- b) 数组是值 (value) 的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。

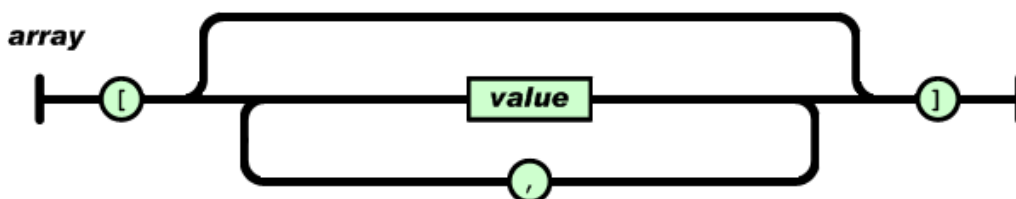


图 2.3 JSON 数组^[14]

如下，名称“患者”对应的值 (value) 为数组 (array)，包含在“[”与“]”之间，表示一组患者：

```
{
  "patient": [
    {

```



```
"name": "张三",  
  "age": 24  
},  
{  
  "name": "李四",  
  "age": 25  
}  
]
```

- c) 值 (value) 可以是双引号括起来的字符串 (string)、数值 (number)、true、false、null、对象 (object) 或者数组 (array)。这些结构可以嵌套。

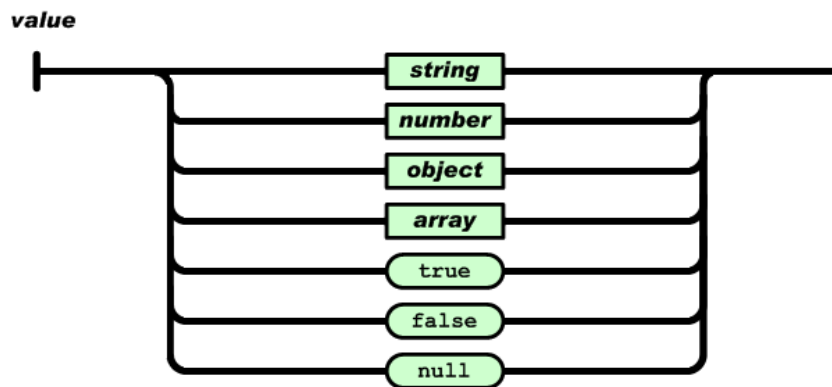


图 2.4 JSON 值^[14]

各类型数据的 JSON 格式如下所示：

```
{  
  "array": [  
    1,  
    2,  
    3  
  ],  
  "boolean": true,  
  "null": null,  
  "number": 123,  
  "object": {  
    "a": "b",  
    "c": "d",  
  }  
}
```

```

    "e": "f"
  },
  "string": "Hello World"
}

```

- d) 字符串（string）是由双引号包围的任意数量 Unicode 字符的集合，使用反斜线转义。一个字符（character）即一个单独的字符串（character string）。字符串（string）与 C 或者 java 的字符串非常相似。

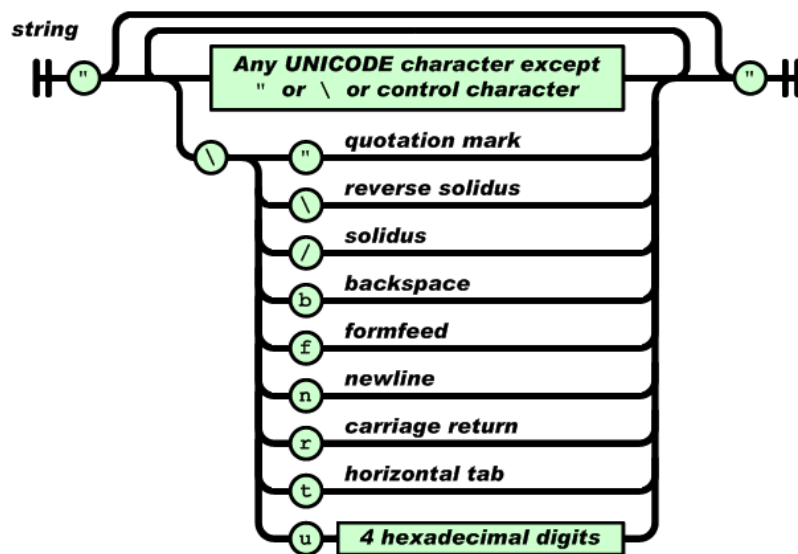


图 2.5 JSON 字符串^[14]

- e) 数值（number）也与 C 或者 java 的数值非常相似。除去未曾使用的八进制与十六进制格式。除去一些编码细节。

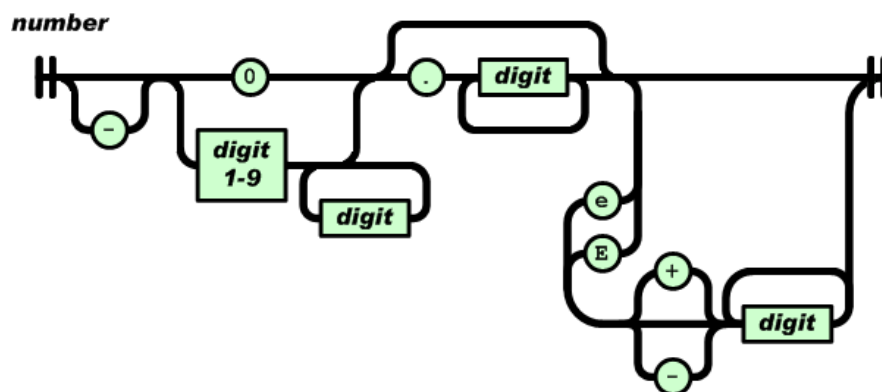


图 2.6 JSON 数值^[14]

2) JSON 的序列化与反序列化

序列化（Serialization）是将对象状态转换为可保持或传输的格式的过程。与序列化相反的就是反序列化，它将流转换为对象。通过 JSON 的序列化和反序列化可轻松实现对象的转换和传递。在 .NET 中，提供了一套 .NET 类型与 JSON 类型转换的标准^[16]，见表 2.1。

表 2.1 .NET 类型与 JSON 类型的映射^[16]

.NET 类型	JSON 类型	注释
所有数值类型，例如 <code>Int32</code> 、 <code>Decimal</code> 或 <code>Double</code>	<code>Number</code>	不支持 <code>Double.NaN</code> 、 <code>Double.PositiveInfinity</code> 和 <code>Double.NegativeInfinity</code> 等特殊值，他们会导致无效的 JSON
<code>Enum</code>	<code>Number</code>	
<code>Boolean</code>	<code>Boolean</code>	
<code>String.Char</code>	<code>String</code>	
<code>T:System.Timespan,Guid,Uri</code>	<code>String</code>	这些类型在 JSON 与 XML 中的格式相同（基本上， <code>TimeSpan</code> 采用 ISO 8601 持续时间格式， <code>GUID</code> 采用 “12345678-ABCD-ABCD-ABCD-1234567890AB” 格式， <code>URI</code> 则采用其本来的字符串形式，例如 “ <code>http://www.example.com</code> ”）。
<code>XmlQualifiedName</code>	字符串	格式为 “名称:命名空间”（第一个冒号之前的所有内容都是名称）。可以缺少名称或命名空间。如果没有命名空间，则也可以省略冒号。
字节类型数组	数字数组	每个数字都表示一个字节的值
<code>T:System.Datetime</code>	<code>Datetime</code> 或 <code>String</code>	
<code>T:System.DateTimeOffset</code>	复杂类型	
XML 和 ADO.NET 类型	<code>String</code>	
集合、字典和数组	数组	
复杂类型（应用了 <code>DataContractAttribute</code> 或 <code>SerializableAttribute</code> ）	复杂类型	数据成员变为 JavaScript 复杂类型的成员
实现 <code>ISerializable</code> 接口的类型	复杂类型	与其它复杂类型形同，但不支持某些 <code>ISerializable</code> 类型
任何类型的 <code>Null</code> 值	<code>Null</code>	也支持可以为 <code>null</code> 的类型，这些类型映射到 JSON 的方式与不可以为 <code>null</code> 的类型相同。

2.2.2 openEHR 规范介绍

openEHR 规范是由 openEHR 组织^[17]制定的一套开放的电子病历规范，其目的是将医疗领域知识从具体的临床信息中分离出来，并建立两层模型——参考模型（Reference Model, RM）和原型模型（Archetype Model, AM）^[18]。参考模型是一组表达医学知识和概念的通用基础数据类型和数据结构，通过对参考模型添加约束的方式来定义原型，大部分的原型都是从少数几个固定参考模型类派生出来的。医疗信息系统的开发主要基于参考模型，领域知识由原型定义，这就有效地降低了系统对领域知识的依赖，从而使系统能够适应领域知识的变化^[18, 19]。openEHR 规范的目标是实现 EHR 系统内部以及 EHR 系统之间的健康信息共享，由 openEHR 机构专门负责制定。

openEHR 官方公布了一种属性+路径（node + path）方法^[12, 19, 20]，在原型中，每个原型属性（node）具有唯一的路径（path）。openEHR 提出了一种原型数据查询语言（Archetype Query Language, AQL）^[12, 21]，用于原型数据查询，如表 2.2 所示。AQL 的一个主要特性是它与系统平台无关，独立于具体应用、编程语言、系统环境和数据存储^[19]。

表 2.2 原型数据查询语言

子句	关键字	参数
SELECT	SELECT	Attribute identify path in archetype
	FROM	Archetype name
	WHERE	Attribute identify path in archetype operator(>, >=, <, <=, !=) condition value
	ORDER BY	Attribute identify path in archetype

原型查询返回值以 dADL(Data ADL)^[12]形式展现。dADL 数据是一种任何机器都能理解的，如下所示：

```

person = (List<PERSON>) <
  [01234] = <
    name = <
      forenames = <"Sherlock">
      family_name = <"Holmes">
      salutation = <"Mr">
    >
    address = <
      habitation_number = <"221B">
      street_name = <"Baker St">
      city = <"London">
      country = <"England">
    >
  >
  [01235] = <
    -- etc
  >
>

```

另外，dADL 是一种包含语意的数据标记结构，对于大数据量的数据，dADL 也存在着一定的效率低下问题，尤其是在大数据量 dADL 进行网络传输的情况下。在实际开发中，可以通过损失一定的语意，用一种更加轻巧的数据来表示 dADL，实现数据传输效率的提升。JSON 是一个很不错的选择，它实现复杂数据类型的结构，与 dADL 的结构有很大相似性。

2.3 数据绑定机制

2.3.1 数据绑定原理

数据绑定思想是一种比较常见的软件开发方法，它为应用程序提供了一种简单而一致的方法来显示数据以及数据交互^[22]。例如 Microsoft 的 WPF 框架中提供的数据绑定机制，支持数据绑定的各种属性、灵活的数据 UI 表示形式，以及业务逻辑与 UI 的完全分离^[23]。

数据绑定是在应用程序 UI 与业务逻辑之间建立连接的过程。如果绑定具有正确设置并且数据提供正确通知，则当数据更改其值时，绑定到数据的元素会自动反映更改。数据绑定可能还意味着如果元素中数据的外部表现形式发生更改，则基础数据可以自动更新以反映更改^[23]。例如，如果用户编辑 TextBox 元素中的值，基础数据值会自动更新以反映该更改。

数据绑定的模型可以用如图 2.7 形式来表示，不论要绑定什么元素，不论数据源的特性是什么，每个绑定都遵循该模型。数据绑定实质上是绑定目标与绑定源之间的桥梁，数据源在系统运行时动态绑定到绑定目标。



图 2.7 数据绑定模型^[23]

数据绑定的一种典型用法是将服务器或本地配置数据放置到窗体或其他 UI 控件中^[23]。在 Microsoft 的 WPF 中，此概念得到扩展，包括了大量属性与各种数据源的绑定。在 WPF 中，元素的依赖项属性可以绑定到 CLR（Common Language Runtime，公共语言运行时）对象（包括 ADO.NET 对象等）和 XML 数据^[23]。在 WPF 的数据绑定机制中，数据流的是双向的，正如上文所示，绑定的数据流可以从数据目标流向数据源（例如，当用户编辑 TextBox 的值时，数据源值会发生更改）和从绑定源流向绑定目标（例如，TextBox

内容会随绑定源中的更改而进行更新)。在.NET 中提供了一种属性更改通知机制 (如 INotifyPropertyChanged), 保证了数据绑定中数据流双向流动的实现。

2.3.2 基于 openEHR 原型的数据绑定方法

本系统为数据浏览工具, 不存在数据操作的功能, 所以本系统数据绑定的数据流是单向的, 即从数据源到绑定目标。本系统数据绑定方法的目的是避免频繁的数据库接口开发, 直接实现 UI 层与业务层数据的绑定, 在基于 openEHR 标准的系统中, 原型即为临床业务的表达, 基于 openEHR 标准进行系统开发的优势在于, 领域专家遵照 openEHR 标准制定原型实现临床业务抽象建模, 软件开发人员只需要通过原型来操作数据即可实现临床业务功能开发的工作。

openEHR 原型是基于 openEHR 参考模型设计的业务抽象层模型, 它遵从 openEHR 标准, 是对临床业务的抽象表示。openEHR 两层模型很好地实现了应用层与底层存储的分离, 体现了领域驱动软件设计 (DDD) 的思想。系统中新的医疗数据的产生伴随着新的原型产生, 实现新增医疗数据的可视化就是实现新增原型的可视化。以原型为标准来驱动新增医疗数据的可视化扩展, 可以规范系统扩展。临床产生新的医疗数据后, 可以基于 openEHR 标准建立新增数据的原型, 然后基于新增原型订制开发可视化组件, 即可实现新增医疗数据的可视化扩展。

根据 openEHR 官方标准, 通过原型查询语言 (AQL) 并访问原型数据接口可获得原型对应的数据 (dADL 数据)。基于 openEHR 原型的数据绑定的本质是 dADL 数据与绑定目标的绑定, 数据绑定模型如图 2.8 所示。绑定目标首先需要与原型关联, 同时对外开放 JSON 数据通道, 系统运行时根据绑定目标关联的 openEHR 原型自动访问原型数据接口获得原型数据 (dADL), dADL 映射为 JSON 数据然后绑定到绑定目标, 便可实现整个绑定过程。

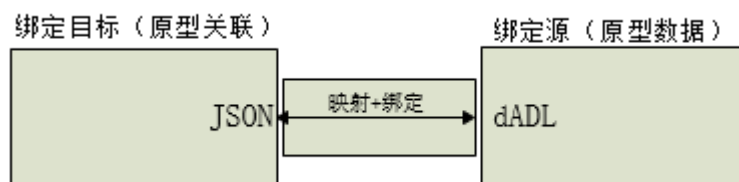


图 2.8 基于 openEHR 原型的数据绑定模型

基于 openEHR 原型的数据绑定的可视化过程如图 2.9 所示, 首先, 将原型查询条件和可视化组件所绑定原型进行组装产生原型查询语言 AQL, 然后通过调用原型访问接口获得查询结果的 dADL 数据对象, 再通过 dADL 映射为 JSON 格式数据, 组件通过 JSON 解

析实现数据的可视化。这种依据原型所订制开发的可视化组件称为原型绑定可视化组件。

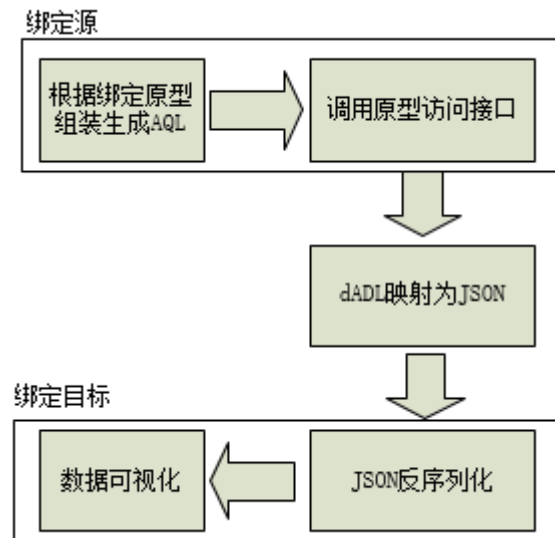


图 2.9 原型绑定模块的可视化过程

在本论文所设计的系统中，绑定目标指的是系统扩展模块，且可绑定多个原型，原型访问接口返回的 dADL 数据需要转换为 JSON 数据。根据以上分析，实现基于 openEHR 原型的数据绑定，需要实现如下几个关键点：

- 实现原型数据（dADL）访问。
- 实现绑定目标与原型的关联。
- 实现 dADL 到 JSON 的映射。

原型数据访问通过 openEHR 官方提供的原型访问接口来实现。绑定目标与原型的关联以及 dADL 到 JSON 的映射通过外置 XML 来实现配置，文件内容如下所示：

```

<Configuration>
  <PrototypeBind PrototypeName="openEHR-EHR-OBSERVATION.lab_test.v1">
    <PropertyBind jsonProperty="VisitID"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0006]/value/value"
Meaning="patient visit id"/>
    <PropertyBind jsonProperty="PatID"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0005]/value/value"
Meaning="patient id"/>
    <PropertyBind jsonProperty="VisitType"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0005]/value/value"
  
```

```

Meaning="visit type"/>

    <PropertyBind jsonProperty="LabTestID"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value/value"
Meaning="labtest id"/>

    <PropertyBind jsonProperty="subject"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0015]/value/value"
Meaning="test subject"/>

    <PropertyBind jsonProperty="orderDept"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0010]/value/value"
Meaning="order department"/>

    <PropertyBind jsonProperty="orderDoctor"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0012]/value/value"
Meaning="order doctor"/>

    <PropertyBind jsonProperty="reqDateTime"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0009]/value/value"
Meaning="order datetime"/>

    <PropertyBind jsonProperty="testCause"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0016]/value/value"
Meaning="cause of test"/>

    <PropertyBind jsonProperty="clinicDiag"
path="/data[at0001]/events[at0002]/data[at0003]/items[at0017]/value/value"
Meaning="clinical diagnose"/>

</PrototypeBind>

```

该 XML 文件为检验原型 dADL 格式数据与 JSON 格式数据的映射配置文件，文件中的 PrototypeBind 节点用于配置绑定目标关联的原型名称，其子节点 PropertyBind 指定原型数据（dADL）的 path 与 json 对象属性成员的映射关系，其中 Meaning 用来保留原型语意，便于人工阅读。通过该配置文件，可以实现 dADL 数据到 JSON 数据的映射，从而进一步通过 JSON 反序列化生成 .NET 对象实现数据可视化。通过该 XML 文件可以获得扩展模块所关联原型的名称，同时根据该文件也可以将原型数据接口返回的 dADL 数据映射为

JSON 数据。另外，当 openEHR 原型发生改动时，只需要更改 dADL 与 JSON 的映射配置文件即可。

2.4 本章小结

本章设计了一种基于 openEHR 原型的数据绑定方法。论文介绍了 openEHR 的两层模型——参考模型（Reference Model, RM）和原型模型（Archetype Model, AM）。通过建立原型实现对医疗领域知识建模，以原型为中心，介绍了原型查询语言 AQL 以及查询返回结果 dADL 的结构。最后，以 openEHR 原型为依据，提出了一种基于 openEHR 原型的数据绑定方法，实现数据的可视化。

基于 openEHR 原型的数据绑定的数据可视化的过程如下所示：

- 1) 根据条件及关联的原型组装原型查询语言 AQL。
- 2) 调用原型访问接口获得查询结果的 dADL 对象。
- 3) 将 dADL 格式数据映射为 JSON 格式数据。
- 4) 绑定目标通过 JSON 解析实现数据的可视化。

3. 系统动态扩展框架设计

本章对系统的动态扩展框架进行设计。首先介绍了本文所研发的医疗数据集成可视化系统的总体结构，引出动态扩展具体需求；然后通过分析对比几种动态扩展技术，确定采用插件技术来实现系统的动态扩展；最后对系统的基于插件的动态扩展技术进行了设计。

3.1 系统总体结构

根据前面分析，本系统需要实现新增医疗数据的可视化扩展，同时需要满足不同临床场景的集成可视化样式的扩展。

系统总体结构如图 3.1 所示，总共分为三层：数据层、系统层和应用层。

1) 数据层

基于 openEHR 两层模型对外提供数据访问服务。系统基于 openEHR 参考模型实现底层数据存储，通过原型模型实现数据的业务抽象，基于 openEHR 官方所提供的原型查询语言（AQL）开放通用原型访问接口，原型数据的访问方式类似于关系型数据库中的 SQL。

浙江大学的王利、闵令通等提出了一种基于 openEHR 的原型关系映射方法^[24]，将原型映射为关系数据表、原型属性映射为关系数据表字段，实现了原型数据的底层持久化，他们基于以上研究构建了高度可扩展的临床数据中心（CDR），通过原型查询语言实现对原型数据的操作（增、删、改、查）。本论文在其工作基础上实现对系统层的设计与开发。

2) 系统层

负责实现系统扩展的框架，提供插件系统，包括系统终端、集成视图库和可视化组件库。

系统终端为用户浏览终端，负责与用户交互。集成视图是指满足不同临床场景的数据集成浏览的视图，通过一定的数据导航方式在一个屏幕下对多数据进行集成展示。可视化组件负责对特定种类医疗数据的可视化，其分为两种，一种是原型订制可视化组件，实现基于 openEHR 原型的数据绑定功能；另一种是非原型订制可视化组件。

3) 应用层

在系统层的基础上根据用户的个人订制生成上层应用视图。在不同的临床场景，

对数据的可视化需求可能不同，所以系统应用需要表现出一定的个性化。用户根据个人的可视化需求或喜好订制特定的集成视图，实现数据的集成可视化。

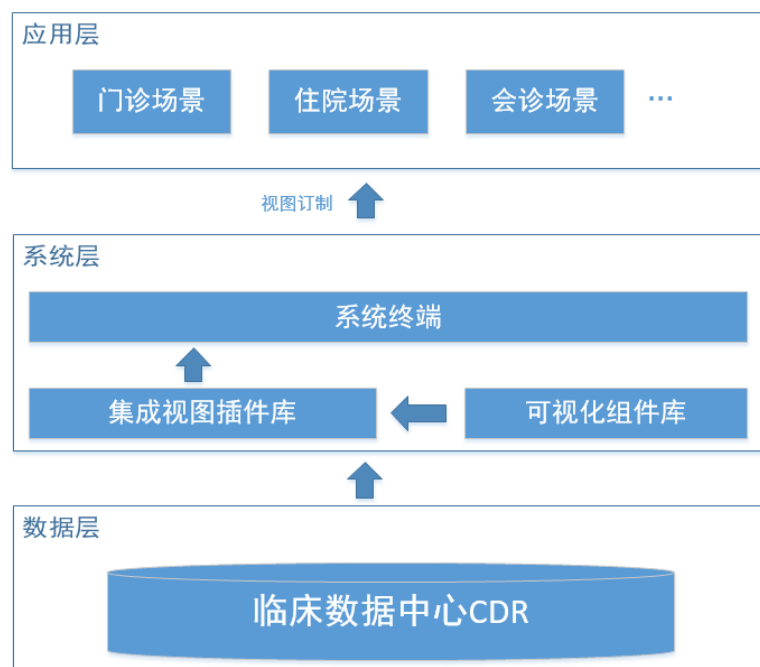


图 3.1 系统层次结构

系统动态扩展表现在系统层的集成视图的动态扩展和可视化组件的动态扩展，后面几节将对系统的动态扩展框架进行分析和设计。

3.2 动态扩展技术研究

软件工程发展到今天，软件重用一直是困扰软件工作者们的一个话题^[25]，从一开始的结构化编程，面向对象编程，再到 COM 编程，目标只有一个，就是希望软件能像堆积木一样组装起来^[26]。软件的动态扩展技术也是从这个目标出发，通过功能抽象，随着业务或需求的变更能实现动态的功能扩展或替换。

本论文中的系统为基于 .NET 平台的 C/S 架构应用程序，下面主要对 .NET 平台所涉及的几个动态扩展技术作论述与对比，并选择适合本系统的一种技术。

3.2.1 COM 组件技术

COM (Component Object Model, 组件对象模型)^[27]组件是微软公司为了计算机工业的软件生成更加符合人类的行为方式而开发的一种软件开发技术。在 COM 架构下，人们可以开发出各种各样的功能专一的组件，然后将它们按照需要组合起来，构成复杂的应用系统^[28]。COM 组件需要遵循 COM 规范编写，COM 规范就是一套为组件架构设置标准的

文档。COM 的基本出发点是让某个软件通过一个通用的机构为另一个软件提供服务。

COM 的前身是 OLE (Object Linking and Embedding, 对象链接和嵌入), 它是 OLE 的 Microsoft 版本。OLE 的第一个版本使用动态数据交换 (Dynamic Data Exchange, DDE) 作为客户及组件之间的通信方式。OLE1 中并没有引入 COM。但是 DDE 非常缓慢, 而且效率不高。其实 OLE 是 Microsoft 的复合文档技术, 它的最初版本只是瞄准复合文档, 在后续版本 OLE2 中导入了 COM。由此可见, COM 是应 OLE 的需求诞生的, 它的第一个使用者是 OLE2, 所以 COM 与复合文档间并没有多大的关系。然而 COM 并不是产品, 它需要一个商标名称, 而那时 Microsoft 已经选用了 OLE 作为商标名称, 所以使用 COM 技术的都贴上了 OLE 的标签, 虽然这些技术中大多数都和复合文档没有多大关系。Microsoft 的这一做法让人产生了 OLE 是单指复合文档还是不单指复合文档的疑问^[29]。其实 OLE 是 COM 的商标, 自然不仅仅指复合文档, 但 Microsoft 很难或者要花费很大精力去解释清楚, 因此后来选择改用 ActiveX 作为 COM 新的商标, ActiveX 指宽松定义的、基于 COM 的技术集合, 而 OLE 仍然仅指复合文档^[29]。COM 是一个说明如何建立可动态交替更新组件的规范, 它提供了客户和组件为保证能够互操作应该遵循的标准。该标准对于组件架构的重要性同其他任何一个具有可交替更新部分的系统是一样的。

COM 规范的定义没有依赖特定的语言, 开发组件对象所使用的语言与调用组件的客户端程序可以采用不同的编程语言, 只要组件符合 COM 规范即可。在 Windows 平台下, 我们一般将 ActiveX 控件作为 COM 组件的代名词, ActiveX 是 Microsoft 为抗衡 JAVA 技术中而提出的, 此控件的功能和 java applet 功能类似。在 Windows 平台下, ActiveX 控件提供了一种类似于 dll 动态链接库的调用, 不过它与普通 dll 的唯一区别是 ActiveX 控件不注册不能被系统识别并使用。ActiveX 控件的发布文件一般以 ocx 或 dll 为后缀名。

在 .NET 平台下使用 COM 组件是通过 .NET 平台所提供的称作 COM Interop 互操作技术来实现的, 该技术支持在托管代码中使用 COM 对象。

3.2.2 .NET 插件技术

在 .NET 中, 插件机制一般都是基于 .NET 的反射机制, 宿主程序运行时反向加载满足系统接口规范的程序集, 实现插件的运行时加载, 同时在 .NET 平台下, 官方也提供了一些插件框架, 如 MEF 和 MAF。

1) 反射机制实现插件框架

系统要实现插件机制, 需要对外提供接口, 宿主系统在加载插件时, 需要通过 .NET

反射机制判断插件是否满足接口标准及插件实例化等操作。

.NET 反射是指审查元数据并收集关于它的类型信息。元数据是一种二进制信息，用以对存储在公共语言运行时的可移植可执行文件或存储在内存中的程序进行的描述^[30, 31]。在.NET 平台下，插件是以 exe 或 dll 文件的形式发布的，通过反向加载文件可以得到程序集，然后可得到程序集中所包含的模块，通过模块又可以得到每个模块中所包含的类型（Type），其中类型包含了程序集中具体的类、枚举或结构体等的信息。通过类型（Type）可进一步获得类中包含的方法、事件、属性、字段等。图 3.2 为反射示意图，可见，通过反射可以反向获得并操作程序集中的类及类中的方法、属性等信息。通过.NET 提供的如下方法可以通过类型（Type）获得其对象实例，该对象实例实现了系统接口，通过面向对象语言中的多态特性可以对插件进行操作。

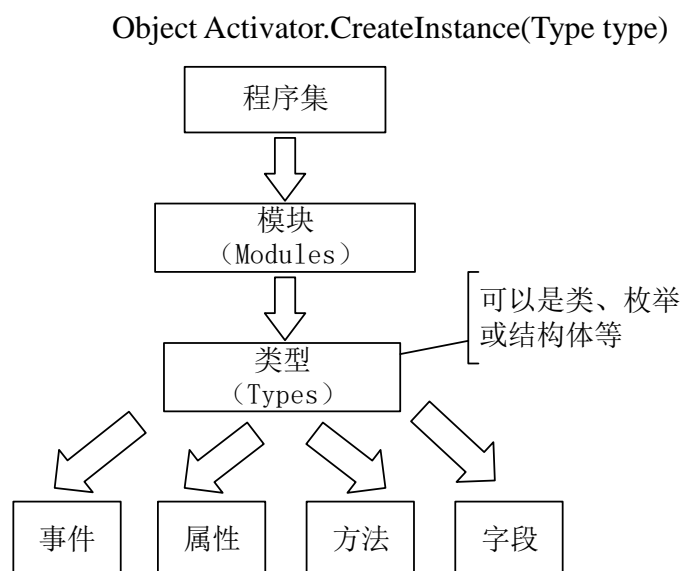


图 3.2 反射机制示意图

2) MEF 框架

MEF（Managed Extensibility Framework，托管扩展性框架）^[32]是.NET Framework 4.0 引入的一个扩展性管理框架，它是官方提供的基于.NET 反射机制实现的一个动态扩展框架。MEF 为开发人员提供了一个工具，让我们可以轻松地实现在不更改原有代码的情况下对应用程序进行扩展，开发人员在开发过程中根据功能要求定义一些扩展点，之后就可以使用扩展点进行扩展；MEF 让应用程序与扩展程序之间不产生直接的依赖，这样就允许在多个具有同样的扩展点的宿主之间共享扩展程序。

MEF 几个基本概念^[32]：

➤ 可组合的部件（Part）：一个部件向其他部件提供服务，并使用其他部件提供的服

务^[33]。MEF 中的部件可以来自任何位置。

- 导出 (Export): 导出是部件提供的服务。某个部件提供一个导出时, 称为该部件导出该服务^[33]。虽然大多数部件只提供一个导出, 但也有部件可提供多个导出。
- 导入 (Import): 导入是部件使用的服务。某个部件使用一个导入时, 称为该部件导入该服务^[33]。部件可导入一个服务也可以导入多个服务。
- 组合: 部件由 MEF 组合, MEF 将部件实例化, 然后使导出与导入程序相匹配^[33]。

典型的 MEF 构建的组合过程:

- 创建组件目录, 即插件的存放目录。
- 创建组合容器 **CompositionContainer**, 组件容器通过组件目录搜索组件的定义。
- 创建一个组件。
- 从组件容器获取其他组件的功能定义, 然后执行匹配组合。

MEF 插件通过使用特性来实现导入和导出。由于 MEF 的部件是以声明方式指定其功能, 因此在运行时可发现这些部件, 应用程序无需硬编码的引用或脆弱的配置文件即可利用相关部件^[32]。通过 MEF, 应用程序可以利用部件的元数据来发现并检查部件, 而不用实例化部件或者甚至不用加载部件的程序集。

3) MAF 框架

MAF(Managed Add-In Framework, 托管外接程序框架)^[34]是 .NET Framework 提供了一种插件式编程模型。这个框架下的插件可以配置为运行在它们自己的应用程序域 (AppDomain)。它最大的特点是可以防止第三方插件引起系统崩溃。

外接程序模型包含一系列的段, 这些段组成负责外接程序和宿主之间所有通信的外接程序管线, 如图 3.3 所示。

外接程序管线

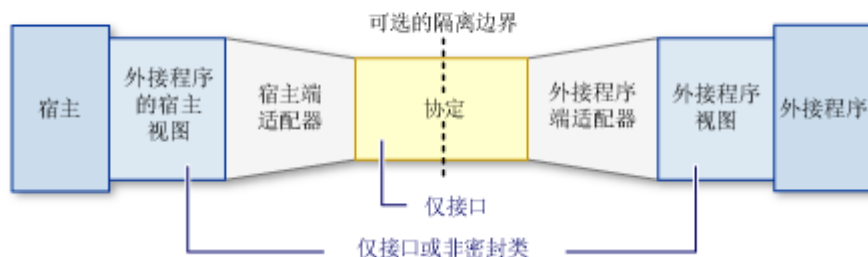


图 3.3 外接程序管线^[34]

管线是在外接程序与外接程序宿主之间交换数据的段的对称通信模型。在宿主和外接程序之间开发这些管线段可以提供必需的抽象层, 用于支持外接程序的版本管理和隔离。

这些管线段的程序集不需要在同一应用程序域中。可以将外接程序加载到自己新的应用程序域或现有的程序域中，甚至加载到宿主的应用程序域中。可以将多个外接程序加载到同一应用程序域中，从而使外接程序共享资源和安全上下文^[34]。

外接程序模型支持并建议在宿主和外接程序之间有一个可选边界，这个可选边界称为隔离边界（也称为远程处理边界），此边界可以是应用程序域或进程边界。管线中间的协定段加载到宿主的应用程序域和外接程序的应用程序域。协定定义宿主和外接程序之间用于互相交换类型的虚方法。若要通过隔离边界进行传递，则类型必须或者是协定类型，或者是可序列化类型。非协定或非可序列化类型必须由管线中的适配器段转换为协议。管线的视图段是抽象基类或接口，用于向宿主和外接程序提供一个它们共享的、由协定定义的方法的视图。^[34]

为了使.NET Framework 发现管线段并激活外接程序，必须将管线段放在指定的目录中^[35]。需要使用指定的目录名。唯一没有指定的名称是管线根目录的名称以及包含外接程序的子目录的名称。所有指定的段名称必须是管线根目录下位于同一级别的子目录。

3.2.3 扩展技术选择

COM 组件方法是传统基于 Windows 开发中比较热门的技术，在.NET 平台下调用 COM 组件，首先需要将 COM 组件注册到 Windows 系统中，然后通过 COM Interop 技术实现.NET 对 COM 组件的调用，COM Interop 对 COM 组件调用的原理是.NET 对 COM 组件的元数据解析，获得 COM 组件中公开的接口，在程序运行时通过代理的方式对 COM 组件进行调用，这里的代理是 RCW（Runtime Callable Wrapper，运行时可调用包）。正是因为 RCW 这个代理的存在，组件使用的很多过程是开发人员不可控的，因此在.NET 平台下基于 COM 组件的扩展技术有很多的局限性，不太适合本系统中的扩展性要求。

.NET 的反射机制很大程度上增大了系统的灵活性，通过反射实现程序集的运行时动态加载，利用系统定义的接口和面向对象编程中的多态特性便可实现对外部程序集约定接口的调用，通过这样一个过程便可实现组件的即插即用的扩展。这便是.NET 插件机制实现的基本原理，.NET 平台下像 MEF 和 MAF 等扩展框架也是依托该原理来实现的。MEF 框架是.NET 平台提供的一种轻量级扩展框架，其关注于使用非常简单的方式来支持具有很强灵活性的可扩展支持，但正是由于其框架太简单，压缩了开发人员的发挥空间，损失了系统设计的灵活性，MEF 框架直接实现了外部程序集到宿主程序中对象的过程，中间的状态无法获得，在本系统中，宿主程序需要获得程序集到对象之间的中间状态来实现更

多的灵活性，如获得类的 **Type** 类型等，所以 **MEF** 框架不适合本系统的扩展性要求。**MAF** 框架关注具有隔离、安全、多版本支持的插件平台架构，但是 **MAF** 框架过于庞大和复杂，尤其是实现插件管道部分，**MAF** 框架对第三方插件开发人员的技术要求较高，不太适合系统的推广，所以 **MAF** 框架也不能满足本系统的设计要求。

综上所述，由于 **COM** 组件的不可控性，本系统不采用 **COM** 组件技术实现扩展。**.NET** 插件技术通过 **.NET** 反射机制实现，可以很好的实现组件的即插即用特性，由于 **.NET** 平台下的 **MEF** 和 **MAF** 框架都有其局限性，均不适合本系统的设计需求。本系统直接采用 **.NET** 反射技术来实现插件机制，并且在参考 **MEF** 和 **MAF** 的各自优点的基础上，使本系统所实现的插件机制具有更高的易用性和隔离性（或安全性）。

3.3 插件式动态扩展框架设计

3.3.1 基于插件的系统扩展框架结构

框架需要实现两种类型的数据扩展，一个是集成视图的扩展，目的是实现对医疗数据导航和集成显示样式的扩展。另一个是可视化组件（插件）扩展，目的是随着临床数据种类的增多，能够快速实现对新增数据可视化扩展。

集成视图作为集成可视化系统的数据呈现形式，起到医疗数据导航和索引的作用，能够让医生快速定位到自己所需要的数据。根据临床业务的不同，不同的临床场景所关注的医疗数据可能不同，数据的导航或索引方式需要满足多样化来适应不同医疗场景的需求和不同医生的喜好。通过开放第三方开发接口的方式来实现集成视图动态扩展。

可视化组件起到数据可视化的作用，是医生所需浏览数据的具体呈现。对于可视化组件，需要满足扩展性和复用性：

- 扩展性。一般情况下随着医疗数据种类的增加或改变，往往伴随着新的数据可视化形式的扩增，因此可视化组件需要满足一定的灵活性和扩展性。对于可视化组件的扩增的实现同样可以通过开放第三方开发接口的方式来实现。
- 复用性。可视化组件的作用是实现具体医疗数据的可视化，为了实现可视化组件的复用，系统需要提供可视化组件的调用与通信机制，即实现集成视图插件对可视化组件的调用和集成视图插件与可视化组件之间的通信。可以通过可视化组件对外开放编程接口的方法来实现，同时可视化组件需要提供编程接口的说明文档。

系统框架需要开放两种接口，一种是集成视图扩展接口，另一种是可视化组件扩展接

口。从架构上来说，系统主要包括四部分内容：系统终端、集成视图插件、可视化组件（插件）和 CDR 数据访问服务，框架如图 3.5 所示。

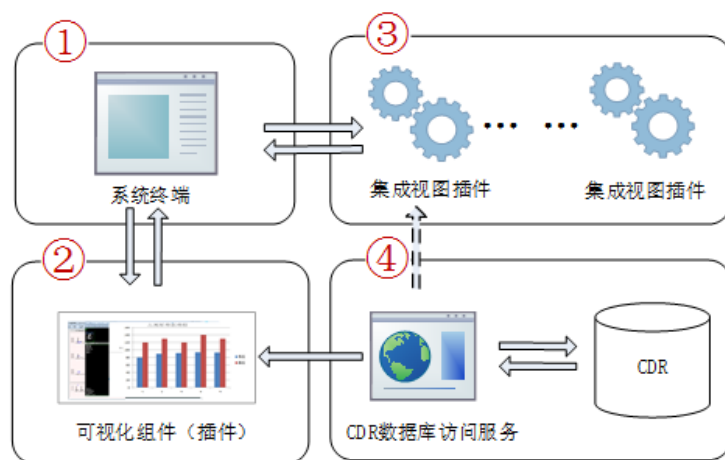


图 3.5 集成可视化系统框架

框架中各部分功能划分及说明如下：

1) 系统终端

系统终端即系统客户端，它是用户直接接触到的软件客户端，用于加载插件和相应插件响应处理，主要作用如下：

- 加载可视化组件插件和集成视图插件。
- 布局显示当前用户订制的视图插件。
- 实现集成视图插件对可视化组件的调用。
- 实现集成视图插件与可视化组件的通信。
- 通过第三方开发接口的形式为集成视图插件提供一些辅助功能，如影像对比功能实现影像对比分析、病人自动关联功能实现相同病人的自动关联、查找相似病症患者功能实现形似病人诊疗方案的对比分析等。

2) 可视化组件（插件）

可视化组件分为两种——原型绑定可视化组件和非原型绑定可视化组件。

负责具体医疗数据的可视化，如医学影像、心电波形等，每个可视化组件（插件）均对应一种数据类型。一种数据类型可对应多个可视化组件（插件），如检查类型对应影像、心电、报告三个可视化组件（插件），分别实现医学影像可视化、心电波形可视化和检查报告可视化。通过可视化组件（插件）第三方开发接口，以插件的方式开发可视化组件，可实现新增医疗数据类型的快速可视化扩展。

可视化组件要实现复用性，则即需要能接收外部参数，也需要能够对外传递内部响应。

第三方开发人员在不知源码的情况下通过可视化组件（插件）所实现接口及其说明文档便可使用该组件实现医疗数据的可视化。

可视化组件插件发布到服务端，系统客户端在启动时会把最新版本的可视化组件更新到本地。

3) 集成视图插件

集成视图通过系统提供的第三方开发接口来实现动态扩展，负责显示病人医疗数据的概要信息，用一定的可视化规则来布局和导航医疗数据。集成视图插件通过调用系统提供的第三方开发接口来获得可视化组件实现具体医疗数据的可视化。

集成视图插件同样发布到服务端，同时客户端提供视图配置功能，允许用户根据个人喜好订制不同的集成视图，实现系统个性化。

4) CDR 数据访问服务

即临床数据仓库（CDR）的数据访问服务接口，用于获取病人医疗数据。系统访问采用基于 openEHR 的数据访问方式，基于原型进行临床数据的查询和获取。

Clever 平台根据 openEHR 官方提供的参考模型，对临床医疗数据作了抽象建模，并实现了原型模板对关系型数据库的映射。Clever 平台数据访问服务，提供了基于原型的查询功能，对外隐藏了底层数据的存储。

3.3.2 系统成员分析

医疗数据集成可视化系统所涉及的人员主要包括四部分：系统设计及开发人员、集成视图插件开发人员、可视化组件（插件）开发人员、系统用户（医生等）。在分析系统的框架之前需要首先清楚各人员的角色及其作用，这样可以更好的理解系统的框架。

- 系统设计与开发人员，负责插件扩展框架的设计与宿主系统开发，负责制定插件之间相互调用和通信的标准。提供插件扩展工具，保证第三方开发人员能轻松实现插件的扩展，保证系统的易用性。
- 组件开发人员，负责可视化组件的开发，使用插件扩展工具实现所开发组件的扩展。可视化组件除了具有扩展性外，还具有复用性，需要满足集成视图插件开发人员调用的需求。在组件扩展的同时提供组件的使用说明文档。
- 视图开发人员，负责集成视图插件的开发，使用插件扩展工具实现视图插件的扩展。在开发视图插件时，开发人员需要根据系统提供的可视化组件使用说明文档来实现具体医疗数据的可视化。在视图扩展的同时，将集成视图的详细说明

文档提供给其他第三方开发人员。

➤ 系统用户，是指系统使用人员，这里特指医院医护人员。

在实际开发过程中，某人可能既是系统设计与开发人员又是组件开发人员同时还是视图开发人员，但从开发模式上我们可以将其看成三个不同的个体。假设本论文后面所提到的不同类型系统人员都是不同的独立个体。

系统人员之间是相互关联的，系统设计和开发人员通过开放接口支持视图开发人员和组件开发人员展开插件的开发；视图开发人员从组件开发人员获取可视化组件实现具体医疗数据的可视化；视图开发人员从系统用户（医护人员）的需求出发设计集成视图并通过系统提供的扩展接口发布给系统用户，系统用户通过订阅的方式来获得特定的集成视图终端。系统各成员的关系可用图 3.4 来表达。其中，视图开发人员和组件开发人员统称为第三方开发人员。

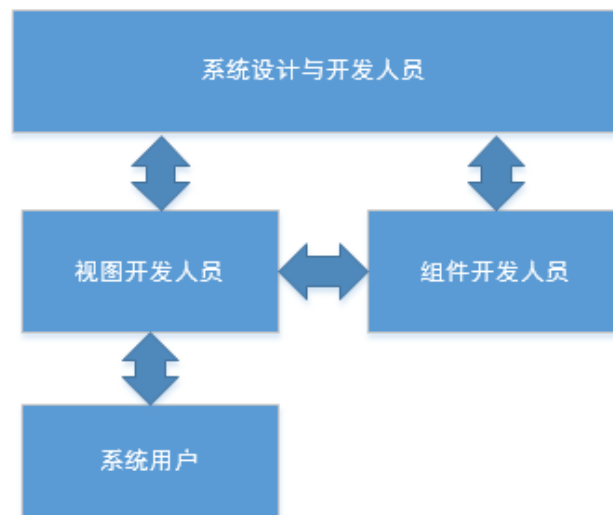


图 3.4 系统人员间关系

3.3.3 插件间通信方法

插件通信主要是指集成视图插件之间、集成视图插件和可视化组件之间、可视化组件之间的通信。插件通信通过相互传送消息的形式实现，消息通信是实现一种灵活系统交互的解决方案，例如在 Windows 中存在消息机制^[36-38]，可实现软件进程内和软件进程间的灵活通信交互。本系统借鉴 windows 的消息机制实现插件通信。

系统对外提供通信接口，如下所示：

```
bool EventSendMessage(IntPtr handle , ICommandMsg)
```

其中，**handle** 代表插件句柄，可唯一标识该插件，代表接收消息的插件，在本系统中，所有插件均继承自.NET 类库中的 **Control** 类，因此可以使用插件句柄作为该插件的唯一标识。。**Msg** 为传递的消息体对象，插件之间的所有交互命令都通过 **Msg** 来传递。如图 3.6 所示为插件通信示意图。

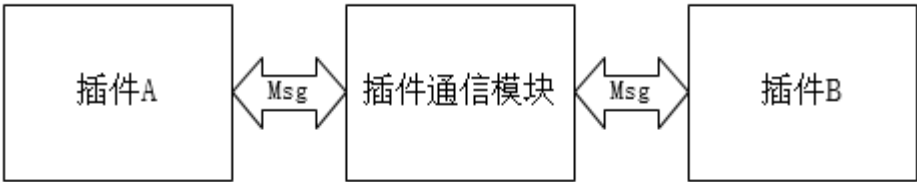


图 3.6 插件通信示意图

插件之间的通信的消息体类型为 **InterfaceCommond**，如表 3.1 所示。其中，**CommandCode** 为命令编码，它唯一标示命令类型，根据命令编码可以对命令参数作特定的解析，实现消息传递；**CmdArgs** 为 **string** 数组，用于传递简单类型的参数（如数值类型、字符类型、**Boolean** 类型等）；**Data** 为 **Object** 类型，用于传输复杂的数据类型，如对象序列化后的 **JSON** 对象数据、图像类型数据等。

表 3.1 消息体数据类型

名称	类型	描述
CommandCode	string	命令编码
CmdArgs	List<string>	参数列表
Data	Object	数据

插件所能处理的消息，保存到插件使用说明文档中，且插件说明文档需要和插件一同发布。

3.3.4 插件接口设计

系统分别通过集成视图扩展接口和可视化组件扩展接口来分别实现集成视图和可视化组件的动态扩展。

3.3.4.1 可视化组件扩展接口

可视化组件包含两种：原型绑定组件和非原型绑定组件。原型绑定组件用于显示特定 **openEHR** 原型所对应的领域知识数据，如医嘱数据、手术数据、病历数据、影像数据、心电图数据等；非原型绑定组件用于实现通用显示样式的扩展，如柱状图组件、饼状图组件等。

可视化组件的扩展接口名称为 **IIVComponent**，其包含内容如表 3.2 所示。

表 3.2 可视化组件接口

名称	类型	描述
ComponentTag	属性	标记，增加组件的灵活性
SetCmd(InterfaceCommand cmd)	方法	返回值 void ，用于接收外部传递命令和参数。
EventSendMessage(IntPtr handle,InterfaceCommand msg)	事件	返回值 Boolean 。用于向指定的插件发送消息。发送成功返回 true ，发送失败或指定插件不存在，返回 false 。
ComponentActionEvent(InterfaceCommand cmd)	事件	用于对外传递内部响应

其中，ComponentTag 用来表示可视化组件标记，目的是增加组件使用的灵活性，允许开发人员（如视图开发人员）对组件进行任何类型的标记。SetCmd 方法用于向可视化组件传递命令和参数，目的是设置可视化组件的可视化效果，其中该方法传递的参数类型是 InterfaceCommond 类型对象。EventSendMessage 为系统开放的消息交互的接口。ComponentActionEvent 事件的作用是对外传递组件内部操作，如组件单击操作等，具体是通过对外传递命令的形式来实现，组件外部通过命令解析来作针对性处理，该事件所传递参数的类型同样为 InterfaceCommond 类型对象。

3.3.4.2 集成视图扩展接口

集成视图的目的是在一个屏幕下集中展示医生所关注的某个病人所有的医疗数据。在视图显示数据前，需要确定当前病人的病人号，同时为了有效利用视图控件，还要确定当前屏幕的分辨率等信息，宿主客户端在调用集成视图插件前，需要首先告诉集成视图当前所在屏幕的大小，宿主客户端选定某个病人或当前病人发生更改的时候，需要能够通知集成视图。

系统提供的集成视图扩展接口的名称为 IPluginView，接口中包含的具体内容如表 3.3 所示。

表 3.3 集成视图接口

名称	类型	描述
ScreenSize	属性	只写属性，用于获得宿主客户端当前屏幕大小。
StartUpPath	属性	只写属性，用于获得宿主进程的运行根目录。
PatientID	属性	只写属性，用于获得宿主客户端当前浏览的病人。
RefreshView()	方法	返回值 void 。外部调用，切换病人时用于刷新视图。
EventGetResource(string strName)	事件	返回值 Control 。用于根据名称获取可视化组件（插件）。
EventSendMessage(IntPtr handle,InterfaceCommand msg)	事件	返回值 Boolean 。用于向指定的插件（可视化组件或其它集成视图插件）发送消息。发送成功返回 true ，发送失败或指定插件不存在，返回 false 。
EventAddDicomContrast(string strExamID,string strItem)	事件	返回值 void 。用于将当前集成视图中的检查记录添加到系统客户端提供的影像对比工具中。
EventShowMessageInfo(string strMsg,string caption,MessageBoxButtons btn,MessageBoxIcon icon)	事件	返回值 void 。用于显示提示信息。提供该接口的目的是保证整个系统提示信息的一致性。

其中，ScreenSize、StartPath 和 PatientID 对外为只写属性，只有宿主客户端才可以改写它们的值，目的是将当前屏幕分辨率、宿主程序根目录和当前病人号等信息告知集成视图。RefreshView()方法用于刷新视图，当集成视图作为当前浏览视图时，该方法会被宿主客户端调用；EventGetResource 事件用于根据名称获得可视化组件实例，它的目的是实现可视化组件的复用，该事件的返回值为 Control 且继承自 IIVComponent，获得组件实例后，根据其说明文档便可实现数据的可视化；EventSendMessage 事件用于向指定的插件发送消息；系统客户端提供了医学影像对比分析的功能，该功能通过 EventAddDicomContrast 事

件对外开放。

3.4 本章小结

本章主要目的是对系统的动态扩展框架进行设计。首先介绍了系统的总体结构，系统需要实现集成视图的动态扩展和可视化组件的动态扩展。

分析介绍了两种动态扩展技术——COM 组件技术和插件技术。结合系统扩展需求及对各扩展技术的对比，最终选择采用反射机制实现系统插件机制的方法实现动态扩展。

本章从集成视图和可视化组件动态扩展的需求出发，对系统框架结构进行了设计。从框架结构上系统主要由四部分组成：系统终端、可视化组件（插件）、集成视图插件、CDR 数据访问服务。

其中可视化组件和集成视图以插件的形式动态扩展，系统终端为插件的宿主程序，三部分通过开放接口或发送消息实现相互通信。同时根据系统扩展需求对插件扩展接口进行了设计。

4. 系统实现与临床实践

本章主要基于前一章的框架设计对系统进行功能设计和实现，并根据临床需求完成可视化组件和集成视图的插件式扩展，最终进行临床实践应用。

4.1 系统模块设计与开发

对于传统框架的软件系统首先需要解决的是软件交付问题，同样本系统也需要设计系统终端和插件的交付方案。系统主要包括插件发布模块、插件更新模块、插件引擎模块和插件监控模块。

4.1.1 插件发布模块

插件发布模块负责将插件发布到服务端，并负责插件版本的在线管理。

1) 发布文件组成

插件在发布时需要包含插件清单文件、客户端图标文件和插件动态链接库文件三种。

➤ 插件清单文件

插件发布时需要提供名为 `manifest.json` 的文件，该文件为 `json` 格式的配置文件，用来配置当前插件版本、名称和依赖文件等信息，集成视图插件的清单文件和可视化组件的清单文件结构略有不同。

集成视图插件的 `manifest.json` 文件的内容如下所示：

```
{
  "Version": "1.0.0.1",
  "Name": "插件",
  "Icon": {
    "Normal_90_30": "Plugin_Normal.png",
    "Clicked_90_30": "Plugin_Clicked.png",
    "Func_23_23": "plugin.png"
  },
  "PluginFile": "PluginView.dll",
  "Dependency": [
```



```
        "DependFile1.dll",  
        "DependFile2.dll"  
    ],  
    "Components": [  
        "Component1",  
        " Component2"  
    ]  
}
```

Version 用来配置当前插件的版本，客户端插件更新时的需要与该属性进行版本对比。**Name** 用来配置当前插件的中文名称，对于集成视图插件，该名称是视图插件在客户端的标示。**Icon** 用来配置系统客户端需要集成视图插件提供的图标文件，插件加载时，这些图标会嵌入到客户端的特定位置。**PluginFile** 用来配置当前插件文件。**Dependency** 用来配置插件文件所依赖的其他动态链接库文件。**Component** 用来配置集成视图运行时所要使用的可视化组件名称。

可视化组件的 **manifest.json** 文件的内容如下所示：

```
{  
    "Version": "1.0.0.1",  
    "Name": "可视化组件",  
    "PluginFile": "VisualizationComponent.dll",  
    "Dependency": [  
        "DependFile1.dll",  
        "DependFile2.dll"  
    ],  
    "Components": [  
        "Component1",  
        " Component2"  
    ],  
}
```

```
}
```

与集成视图清单文件内容不同的是，可视化组件不需要包含图标文件配置项“Icon”，另外，如果可视化组件为原型绑定组件，需要添加原型绑定配置项，即“PrototypeBinding”项，如果为非原型绑定可视化组件，则不需要包括“PrototypeBinding”项。

➤ 图标文件

客户端加载插件时，会将图标文件嵌入到插件宿主客户端的 UI 界面，起到插件的标示作用。该文件内容需要配置在 manifest.json 文件中的“Icon”中。集成视图插件需要图标文件，而可视化组件不需要。

➤ 动态链接库文件

插件动态链接库文件是指插件的发布文件。它又分为插件文件和插件依赖文件两种。插件文件需要满足系统开放的接口标准，即 manifest.json 文件中“PluginFile”对应文件，插件依赖文件是插件文件在运行时需要动态加载的文件，即 manifest.json 文件中“Dependency”所对应文件。

集成视图插件发布文件需要包括上述三种文件，可视化组件（插件）发布文件不需要包括图标文件。插件发布时，需要将图标文件和动态链接库文件打包为 ZIP 压缩包。采用压缩包的形式发布插件的目的是提高插件的网络传输效率。

系统提供插件扩展工具，包括集成视图插件扩展和可视化组件扩展工具，如图 4.1 和图 4.2 所示。通过该工具可以自动生成 manifest.json 文件，并将插件文件打包成 ZIP 文件上传到指定的服务端。



图 4.1 集成视图扩展工具



图 4.2 可视化组件扩展工具

2) 插件发布流程

利用系统提供的插件扩展工具，将清单文件和压缩包文件发布到服务端，实现插件的发布。插件的发布流程如图 4.3 所示。

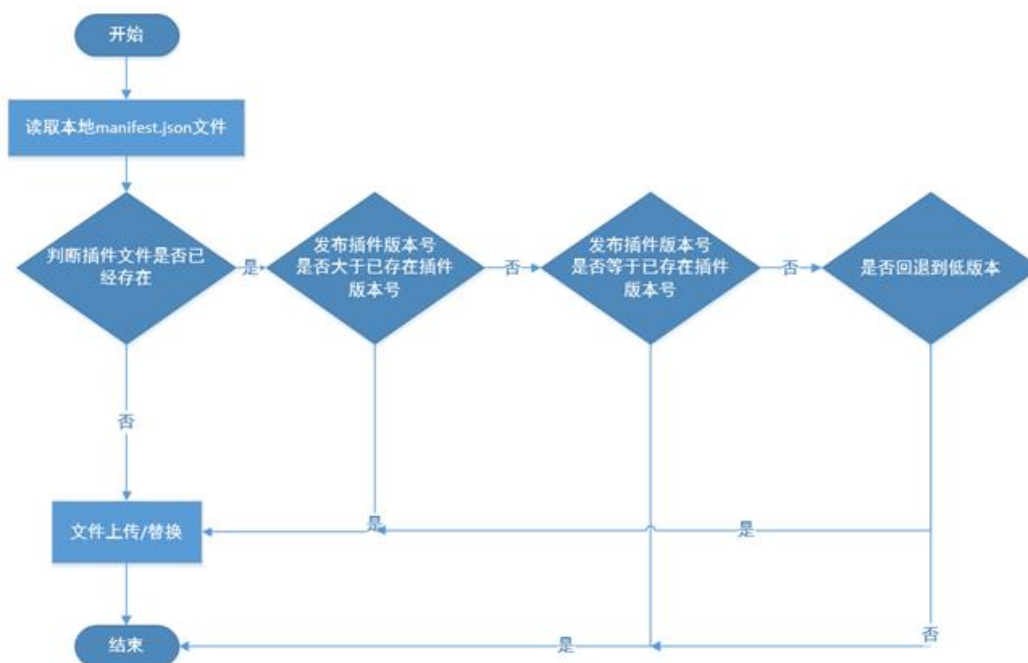


图 4.3 插件发布流程图

首先，根据 manifest.json 清单文件，判断服务端是否存在该插件，如果不存在则直接上传，完成发布，如果已经存在该插件，对比当前发布插件和已存在插件的版本号，如果发布插件的版本号高于已存在插件的版本号，则将已存在插件用当前发布插件替换掉，完成发布，如果发布插件的版本号等于已存在插件版本号，则不将当前发布插件上传，完成

发布,如果发布插件的版本号小于已存在插件的版本号,询问用户是否将插件回退到低版本,如果用户选是,将已存在插件用当前发布插件替换掉,完成发布,否则不将当前发布插件上传,完成发布。

4.1.2 插件更新模块

插件更新模块负责客户端插件的更新,根据用户配置远程下载或更新集成视图插件到本地集成视图插件库,并根据集成视图插件所依赖的可视化组件,动态更新本地可视化组件库。

本文所设计的系统允许医生在客户端订制视图,即配置宿主客户端加载哪些集成视图插件。系统宿主客户端启动时,读取该视图配置文件获得需要加载的视图插件,遍历这些插件并更新本地插件文件,使其与服务端版本保持一致。更新流程如下:

首先判断服务端是否存在配置文件中的视图插件,如果服务端不存在且本地存在该插件,则删除本地插件;如果服务端存在且本地不存在该插件,则直接将服务端插件发布文件下载到本地并解压压缩包;如果服务端存在且本地也存在该插件,判断服务端版本号是否大于本地版本号,如果大于本地版本号,则删除本地插件文件,然后下载服务端插件发布文件到本地并解压压缩包。视图插件更新成功后,根据插件版本清单文件 `manifest.json`,更新该视图插件所依赖的可视化组件,可视化组件的更新流程同视图插件的更新流程。集成视图插件更新流程如图 4.4 所示。

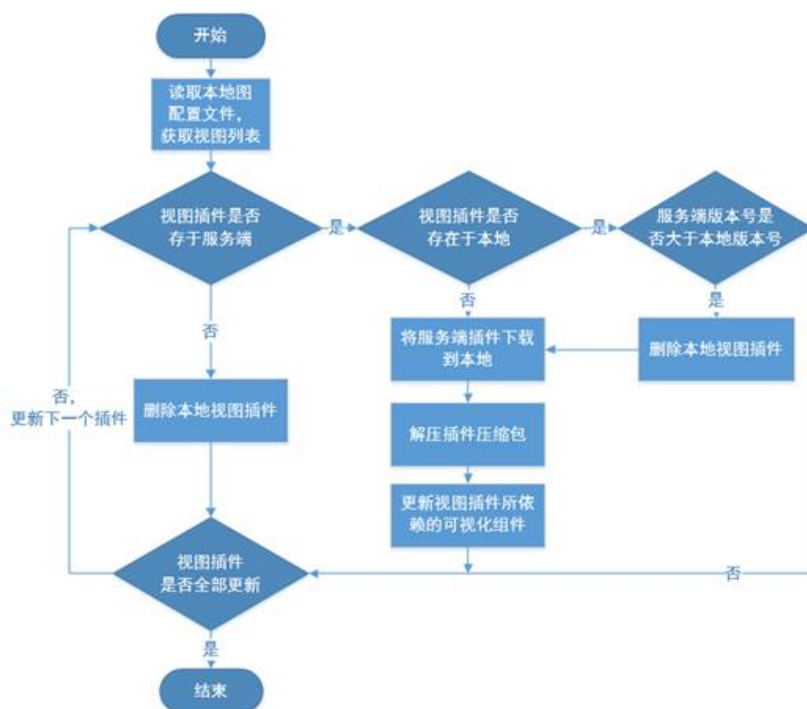


图 4.4 集成视图插件更新流程

4.1.3 插件引擎模块

插件引擎模块负责实现插件机制，实现插件的运行时加载，同时负责插件调度与通信功能。

1) 插件加载

插件加载是指系统在运行时根据插件接口对插件文件的发现和加载，它是插件框架的核心。插件加载包括集成视图插件加载和可视化组件加载两个过程。系统更新模块更新完成后，系统需要将插件加载到宿主客户端中。系统通过遍历集成视图插件文件夹，分别加载各集成视图插件及其所依赖的可视化组件。如图 4.5 所示。

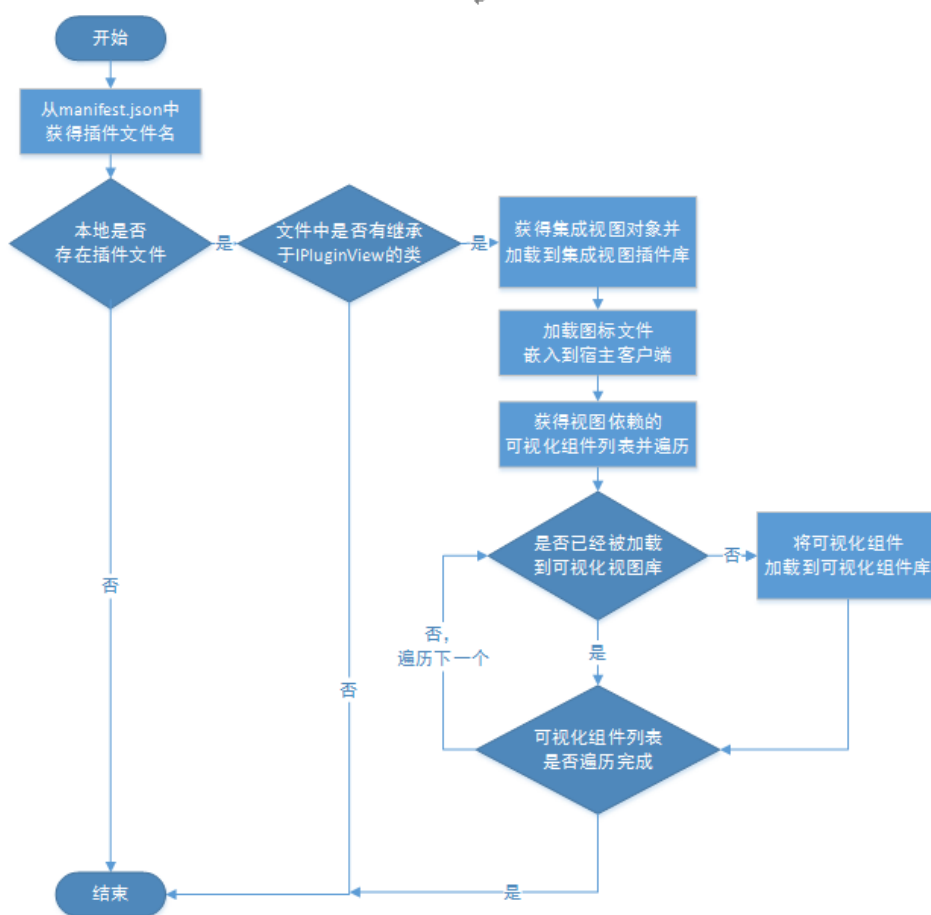


图 4.5 插件加载流程

首先读取 manifest.json 文件中“PluginFile”所配置的插件动态链接库文件名，反射加载程序集，找到继承于 IPluginView 接口的类型，获得该类型对应的对象加入到集成视图插件库，插件视图加载成功后，再根据 manifest.json 文件加载视图所依赖的可视化组件，并加入到可视化组件库。

2) 插件调度与通信

插件加载成功后，在系统中形成了两种插件库——集成视图插件库和可视化组件插件库。其中，集成视图插件库中存放的是各插件视图的对象，直接被系统宿主客户端加载，对于系统宿主客户端来说，每种集成视图只能存在唯一的对象；可视化组件库存放的是可视化组件的类型而不是具体的可视化组件对象，在系统运行时，集成视图插件通过插件引擎模块实现对可视化组件的调用，实现数据的可视化，如图 4.6 所示。

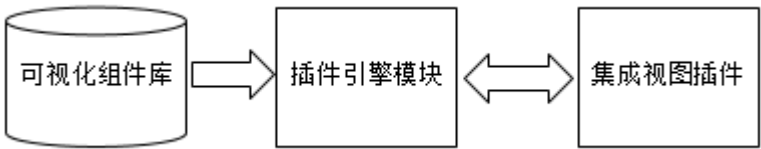


图 4.6 可视化组件调用

该模块同时负责插件消息的分发，当某个插件发起消息通信时，会将消息发送到插件引擎模块，该模块根据句柄将消息发送到目标插件，并调用目标插件的消息处理函数，实现消息的通信过程。

4.1.4 系统监控模块

对于插件系统来说，插件是由第三方开发人员开发的，系统需要保证某个插件的异常不会对系统和其它插件的正常运行造成影响，同时当系统出现异常后，需要记录异常的详细信息，实现通过日志分析来发现系统异常的原因，实现系统运行监控的目的。

系统监控模块主要实现两个作用——系统异常捕获处理和日志记录。该模块的工作流程如图 4.7 所示。

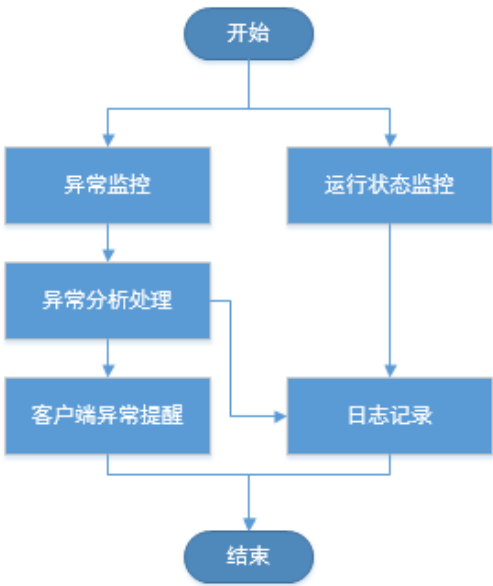


图 4.7 系统监控模块执行流程

系统监控内容主要包括系统运行状态和系统异常。系统运行状态是指开发人员所指定的内容，如变量值的变化等。系统异常是指系统的运行结果偏离了预期结果导致数据无法正常浏览或者系统出现运行时 **Bug** 导致无法按照正常流程继续执行。

当系统发生异常后，首先系统监控模块可以捕获到异常，然后进行异常分析，最后在客户端给出针对性的异常提醒并将异常分析结果保存到日志记录文件中。为了对后续的日志管理和分析，系统监控模块将日志根据重要程度划分为五个等级，对于高级别的日志内容需要上传到服务端保存，如表 4.1 所示。

表 4.1 日志级别分类

异常级别	代码	描述
DEBUG	0	调试日志，保存到本地
INFORMATION	1	信息日志，保存到本地
WARNING	2	警告日志，保存到本地和服务端。
ERROR	3	错误日志，保存到本地和服务端
FATAL	4	严重错误日志，保存到本地和服务端

DEBUG 日志是最低级别的日志，一般来说在系统实际运行过程中是不输出的。该日志级别主要用于系统调试，主要目的是了解系统运行时状态，如变量的值等，系统发布版本中最好不要使用该级别的日志。该级别的日志只需保存本地。

INFORMATION 日志主要用来将系统运行的状态反馈给系统的最终用户，该级别的日志需要具有实际意义，需要用户能够看懂日志的意义。该级别的日志一般作为系统产品的一部分发布给用户。该级别的日志只需保存本地。

WARNING 日志主要用来记录系统运行过程中系统运行的结果和系统期望的结果不一致的问题或明显的运行时错误但不会引起系统崩溃的异常。该级别的日志既要保存到本地也要保存到服务端。

ERROR 主要用来记录系统运行过程中宿主客户端导致的系统崩溃问题。该级别的日志既要保存到本地也要保存到服务端。

FATAL 是最高级别的日志，主要用来记录系统运行过程中插件引起的运行崩溃问题。该级别的日志既要保存到本地也要保存到服务端。

其中 **DEBUG** 和 **INFORMATION** 级别的日志为系统运行状态监控日志，**WARNING**、**ERROR** 和 **FATAL** 级别的日志为系统运行异常日志，表示系统运行过程中出现了不正常的问题，这三种级别的日志需要上传到服务端，用于远程监控分析系统运行过程中的问题。

日志记录以一次系统运行周期（客户端启动到关闭）为单位，系统启动时开始系统监

控，系统正常关闭或异常退出时，系统监控结束并将 **WARNING** 级别以上的日志上传到服务端。

日志文件以 **XML** 格式保存，每条日志结构如下所示：

```
<LogNode ErrorType="FATAL" DateTime="2015/12/10 15:58:57">
    <LogDetail>TimeLineView 无法找打依赖文件 ViewDocument</ LogDetail >
</LogNode>
```

其中 **LogNode** 节点的 **ErrorType** 属性用来配置日志级别，**DateTime** 属性用来记录异常发生的时间。**LogDetail** 子节点用来配置异常的详细信息。

4.2 临床应用实践

基于系统的动态扩展框架，本节对系统进行临床实践开发，包括可视化组件开发和集成视图插件开发，并对系统进行扩展。该系统已在国内某三甲医院门诊全院上线运行，应用状况良好，本节将对该上线版本的可视化功能进行说明。

4.2.1 可视化组件开发

可视化组件分为原型绑定可视化组件和非原型绑定可视化组件。原型绑定可视化组件通过数据绑定方式实现数据可视化；非原型绑定可视化组件通过接收外部消息携带的数据实现可视化。可视化组件需要实现可视化组件扩展接口 **IIVComponent**。

4.2.1.1 原型绑定可视化组件

原型绑定可视化组件是为（一个或多个）原型订制开发的可视化组件。原型绑定后，组件接收到外部传送的可视化消息，可以内部组装原型查询语句并调用原型访问接口获取数据源并实现可视化。

基于数据可视化的需求及已有原型的基础上，本文开发了八种原型绑定可视化组件，如表 4.2 所示。

表 4.2 原型绑定组件插件

可视化组件	绑定原型	描述
医学影像可视化组件	检查数据原型	用于医学影像数据可视化
心电波形可视化组件		用于心电波形数据可视化
检查报告可视化组件	检查报告原型	用于检查报告文件的可视化
检验数据可视化组件	检验数据原型	用于检验数据可视化
手术数据可视化组件	手术原型	用于手术数据的可视化
	手术状态原型	
处方数据可视化组件	处方原型	用于处方数据的可视化
	处方状态原型	
医嘱数据可视化组件	医嘱原型	用于医嘱内容及日志的可视化
	医嘱状态原型	
	医嘱执行单状态原型	
病历数据可视化组件	病历原型	用于病历文档的可视化

可视化组件可与一个或多个原型绑定，对于多原型绑定需要注意原型的绑定顺序。可以通过 **ShowView** 消息实现组件的可视化，**ShowView** 消息的消息体说明如表 4.3 所示。

表 4.3 ShowView 消息体说明

消息参数	参数说明
CommandCode	值为“ShowView”
CmdArgs	空或数据的唯一标识
Data	空或原型查询返回结果的 JSON 格式数据

其中，**CmdArgs** 传递的数据为数据的唯一标示，如检查号、检验号、手术申请号、医嘱号等，**Data** 传递的数据是原型查询语句返回值的 JSON 结构数据，**CmdArgs** 和 **Data** 至少有一个不为空。组件接收到 **ShowView** 消息后，首先判断 **CmdArgs** 是否为空，如果不为空，内部自动根据绑定的原型生成原型查询语句并调用原型访问接口获得数据实现可视化，如果为空，则从 **Data** 中直接获取原型查询返回结果的 JSON 格式数据实现可视化，可视化效果如图 4.8 所示。

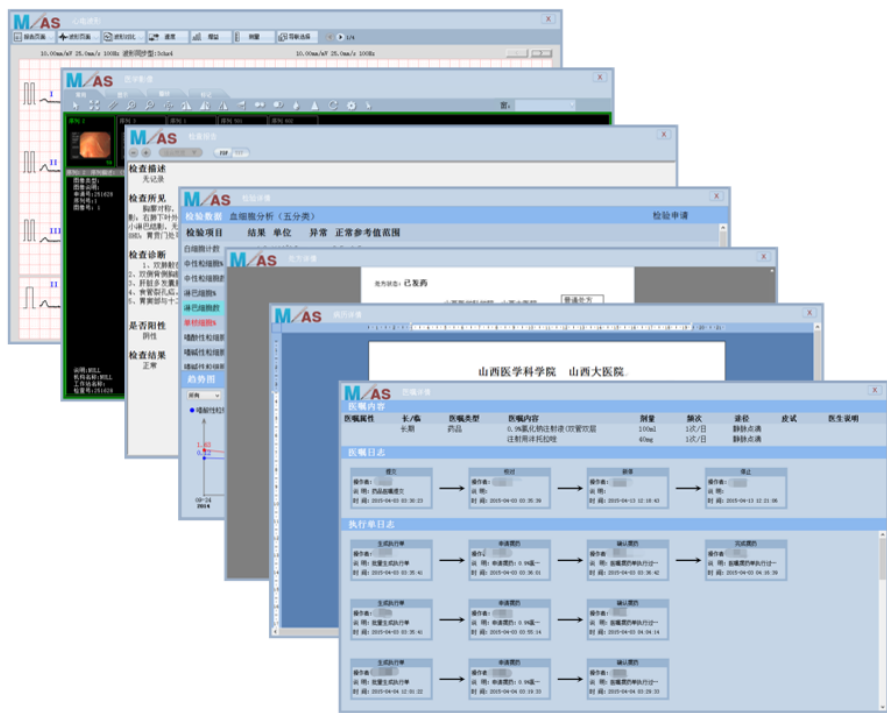


图 4.8 原型绑定可视化组件

原型绑定可视化组件发布时需要包括三种文件：**manifest.json** 文件、组件动态链接库文件和组件依赖动态链接库文件。表 4.2 中组件发布文件中的 **manifest.json** 文件的内容见附录一。

4.2.1.2 非原型绑定可视化组件

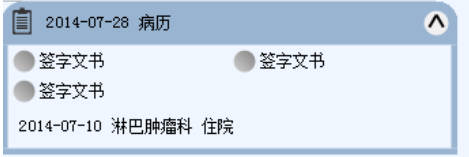
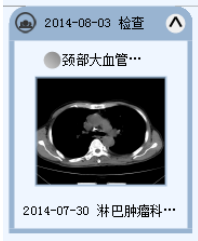
非原型绑定可视化组件是指未与原型绑定的可视化组件，通过接收外部消息传递的数据实现数据的可视化，本文实现了如表 4.4 所示的三种非原型绑定可视化组件。

表 4.4 非原型绑定组件

可视化组件	描述
数据卡组件	用于显示数据记录条目
饼状图组件	用于以饼状图的样式展示数据
柱状图组件	用于以柱状图的样式展示数据

数据卡是数据容器控件，有两种显示外观类型，分表用于显示文本记录数据和图文并茂数据，如表 4.5 所示。其可以接收 **SetDataCardInfo** 和 **FoldDataCard** 两种消息，**SetDataCardInfo** 消息用于向数据卡传递数据，其消息体的 **Data** 用于保存 JSON 对象，控件接收到消息后，通过 JSON 解析实现可视化。**FoldDataCard** 消息用于设置数据卡的折叠和打开，控件的折叠和展开操作通过 **CmdArgs** 来传递。

表 4.5 数据卡组件分类

数据卡类型	类型名称	样式图
普通数据卡	NormalType	
检查数据卡	ImageType	

饼状图组件和柱状图组件显示数据的统计信息，如图 4.9 所示。组件可以接收 SetData 消息，组件显示的数据通过消息体中的 Data 来传递，其传递数据的类型 JSON 对象，组件接收到消息后，通过 JSON 解析实现组件可视化。

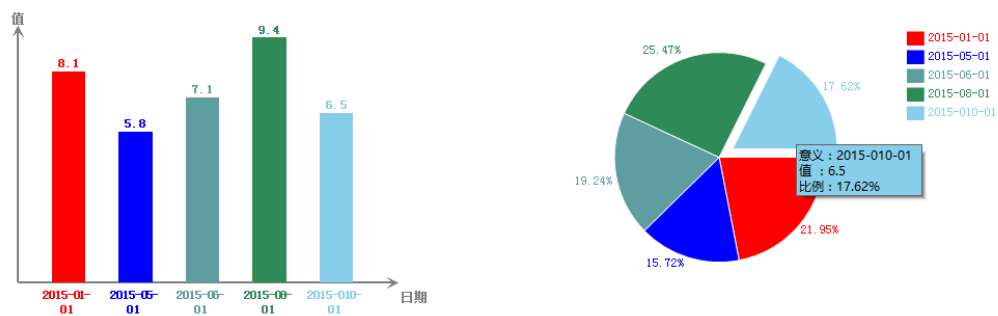


图 4.9 柱状图组件和饼状图组件

4.2.2 集成视图插件开发

集成可视化系统是由不同的集成视图组成，集成视图是用户集成浏览数据的视图。本节将从临床数据可视化的需求出发，基于系统的动态扩展框架，实现了三种集成视图——时间导航视图、就诊导航视图和客观医疗数据视图，客观医疗数据视图又包括检查一览视图和检验一览视图。集成视图插件需要实现集成视图扩展接口 IPluginView。

4.2.2.1 时间导航视图

- 医疗数据具有如下几个特点：
- 时间相关性强。时间是影响医生进行数据浏览的重要因素。
 - 数据分类细致，并且每一类数据往往代表一种临床业务。在不同临床场景下，医生所关注的数据类别往往不同。

根据这两个特点,设计了基于时间轴的分类别数据导航视图,该视图的设计初衷是尽可能在屏幕中显示医生关注的有效信息(即医生所关注数据)。

数据的展示始终以纵向时间轴的形式展现,默认情况下,时间轴上显示病人所有的医疗数据,但提供灵活的数据过滤筛选功能,可通过时间轴的设置来显示特定时间段的指定类别的数据。

在纵向提供两个时间轴,一个是缩略时间轴,一个是详细时间轴,如图 4.10 所示。缩略时间轴上显示的为有医疗数据的年份和月份;详细时间轴上显示的为有医疗数据的时间节点,在时间节点的右侧列出该时间下产生的医疗数据。



图 4.10 时间轴导航视图

该视图只显示医疗数据的概要信息,每种医疗数据对应一种 openEHR 原型,双击数据对应的条目内容调用 EventGetResource 事件获得与对应原型所绑定的可视化组件。通过调用系统提供的消息通信接口可实现集成视图与可视化组件的交互,如让可视化组件显示数据,如图 4.11 所示,医学影像可视化组件显示影像数据。



图 4.11 影像可视化组件

该视图适合对数据要求比较全的临床场景，如会诊场景等；同时视图适合以时间为参考进行数据对比的场景，如门诊场景中病人复诊的情况。

4.2.2.2 就诊导航视图

就诊导航视图是指以病人就诊记录为单位对数据进行组织和导航的视图。一次就诊记录包含了病人一次就诊时间段内的所有医疗数据，即一次门诊或住院期间的数据。

在临床上，一次就诊中的医疗数据可以很好的反应病人的诊疗方案和病情变化，同属一次就诊的不同医疗数据往往具有一定的相关性，例如某次就诊中，病人吃了某种药物后，医生为了查看药物疗效可能会让病人再做某种检查或检验，而检查或检验结果又会决定医生下一步的诊疗方案，那么药物和该检查或检验数据已经之后诊疗计划中产生的医疗数据具有很强的相关性。在临床上，这种相关性很强的数据往往能给临床诊疗带来很大的参考价值。

如图 4.12 所示为就诊导航视图，左侧为就诊导航区域，包括所有就诊记录及就诊记录中包含的数据列表，该列表为树状列表。右侧为数据可视化区域，通过系统开放接口调用可视化组件实现数据详细信息的可视化。



图 4.12 就诊导航视图

该视图适合以一次就诊时间段为单位进行数据浏览的临床场景，如住院查房场景、门诊场景中病人复诊的情况等。

4.2.2.3 客观医疗数据视图

医疗客观数据是指医疗过程中由医疗器械和生化试验等所产生的反应病人健康状况的医疗数据，这些数据不以人的主观认识而改变，具有一定的权威性，在临床上具有重要的作用，也是多数医疗过程中所关注的的数据。

这里所提的客观数据主要包括两种——检查数据和检验数据。医疗检查检验是为临床诊断、疾病治疗和预防以及人体健康提供准确可靠的检测报告的过程，除了医生凭经验诊断外，大多数的临床诊断和辅助诊断信息都来自医疗检查检验^[39]。基于以上分析，决定在系统中提供专门针对检查和检验数据的两个客观数据导航视图——检查一览视图和检验一览视图。

1) 检查一览视图

检查一览视图是对病人检查数据进行导航并集成浏览的视图，如图 4.13 所示，整个视图分为三个区域：

➤ 检查记录导航区域

该区域用于导航检查记录，病人的检查记录全部以缩略图的形式显示在此区域，这部分内容需要第三方开发人员（视图开发人员）设计开发。

➤ 检查报告数据区域

该区域用于显示选定检查记录的检查报告数据，通过调用检查报告原型所对应的

檢查報告可視化組件來實現數據可視化。

➤ 檢查數據區域

該區域用於顯示選定檢查記錄的檢查數據，其中檢查數據包括醫學影像數據和心電波形兩種。通過調用系統接口獲得檢查影像原型所對應的檢查影像可視化組件來實現檢查影像數據的可視化，通過調用系統接口獲得心電原型所對應的心電波形可視化組件來實現心電波形數據的可視化。

在檢查記錄導航區域切換檢查記錄後，可經過系統的插件通信模塊向檢查報告區域中的檢查報告可視化組件和檢查數據區域中的醫學影像可視化組件或心電波形可視化組件發送命令消息，組件通過命令解析，根據數據綁定的情況實現數據可視化。

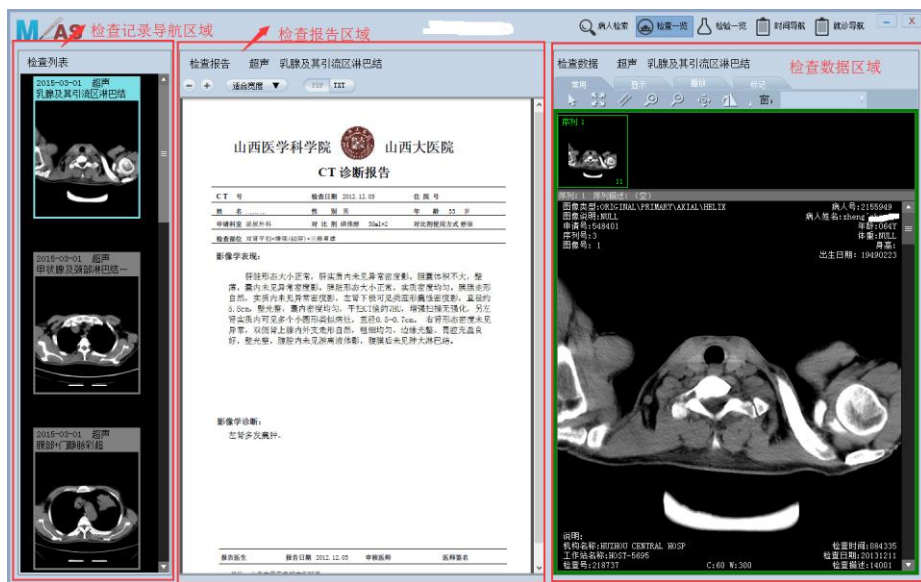


图 4.13 检查一览视图

該視圖是專門針對檢查數據進行集成瀏覽的視圖，適合對檢查數據瀏覽比較頻繁的科室，如骨科門診、骨科住院、放射科報告工作站等。

2) 檢驗一覽視圖

檢驗一覽視圖是對病人檢驗進行導航和集成瀏覽的視圖，如圖 4.14 所示。該視圖在布局上有兩個區域：

➤ 檢驗記錄導航區域

該區域用於對檢驗記錄進行導航，以就診記錄為單位對檢驗記錄進行分組導航。選定檢驗記錄後，可以查看其詳細信息，詳細信息顯示在檢驗結果數據區域。

➤ 檢查結果數據區域

該區域用於顯示某次檢驗的結果數據，同時對於數值型的檢驗項目結果可以在時

间维度上分析其数据走向。通过调用检验原型对应的检验数据可视化组件实现数据的可视化。

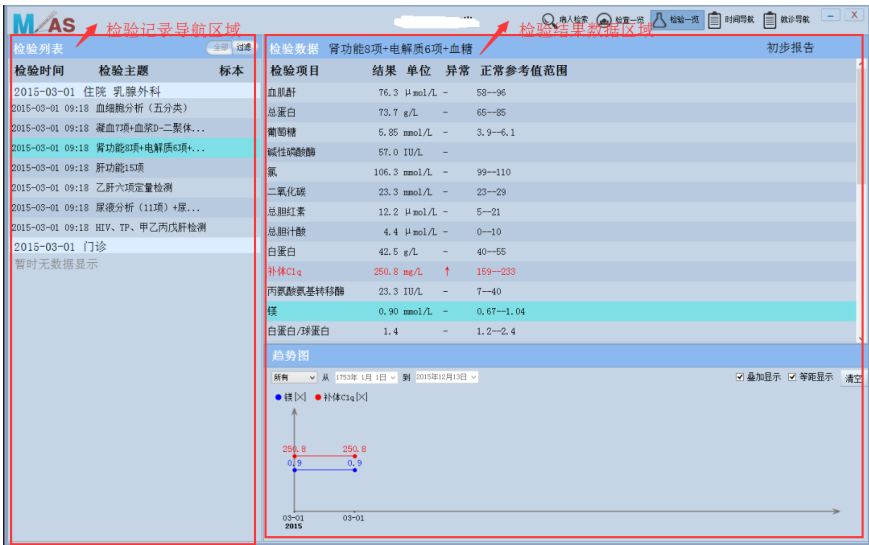


图 4.14 检验一览表视图

该视图是专门针对检验数据进行集成浏览的视图，适合对检验数据浏览比较频繁的科室，如门诊场景等。

4.2.3 视图个性化订制

系统用户可以通过视图订制工具来按需定制自己需要的集成视图，从而实现系统的个性化应用，如图 4.15 所示。



图 4.15 视图订制界面

在该界面进行视图订制后，会在本地生成视图订制配置文件，文件内容如下所示，其中，在 ConfigItem 节点中配置视图插件的名称，保存在 ViewName 属性中。


```
<?xml version="1.0" encoding="UTF-8"?>
</Configuration>
    <ConfigItem ViewName="ExamView"/>
    <ConfigItem ViewName="LabTestView"/>
</Configuration>
```

系统用户可以在视图订制界面根据集成视图的说明来订制满足个人需求的集成视图，系统运行时，会根据视图订制配置文件内容加载视图插件生成上层应用视图，实现系统的个性化应用。

4.3 本章小结

本章对系统各模块的实现和执行过程进行了详细介绍，同时临床应用开发的内容进行了详细介绍。

论文对系统进行分模块设计和实现，系统主要包括插件发布模块、插件更新模块、插件引擎模块和异常监控模块。

论文基于临床场景对医疗数据的可视化需求进行临床应用开发实践，分别从系统可视化组件（包括原型绑定可视化组件和非原型绑定可视化组件）和集成视图的角度实现了系统扩展。系统同时支持视图的个性化订制，实现不同临床场景下医生个性化订制。论文中所开发的系统已在国内某三甲医院门诊全院上线应用，应用状况良好。

5. 总结与展望

5.1 总结

医疗数据集成可视化系统，是对医疗数据进行集成浏览的信息系统，目的是在一个屏幕下通过一定的导航或索引方式对临床医疗数据进行集成浏览。

随着医院信息化的发展，医疗数据的种类和数量不断增多，不同的临床场景对医疗数据的可视化需求不同，集成视图便是在这样的背景下产生的，其作用是满足不同医疗场景对医疗数据的集成浏览需求。医疗数据作为医疗诊断的重要依据，对医疗业务具有极其重要的作用，医疗数据集成可视化系统起到了为临床业务提供数据支撑的作用。

医疗数据种类的不断增多以及临床业务的千差万别要求医疗数据集成可视化系统要有一定的扩展性和灵活性。一方面，医疗数据种类复杂且不断增多，新增种类的医疗数据往往伴随着新的可视化效果，因此系统需要满足新增数据可视化效果的动态扩展；另一方面，临床场景复杂，不同场景对医疗数据的需求以及医疗数据的导航方式的要求不同，因此系统需要满足医疗数据集成视图的动态扩展。

针对动态扩展框架，国内外已经有一定的研究成果，也提出了一些动态扩展方法。对于新增种类的医疗数据，一般采用组件的思想来实现其可视化效果的动态扩展。本论文在调研医疗数据可视化系统动态扩展方法研究成果后，发现存在如下两个问题：

- 系统与底层数据库的耦合程度太高，系统的扩展往往伴随着大量数据库接口的开发。
- 系统各扩展模块之间缺少灵活的通信支持，系统交互能力不足。

这两个问题大大降低系统动态扩展的灵活性和可行性，本论文针对以上问题展开研究，主要内容如下：

- 1) 针对系统与底层数据库高耦合的问题，论文基于 openEHR 两层模型方法实现一种数据绑定机制，避免频繁的数据库接口的开发。分析了 openEHR 规范所提出的系统开发的两层模型方法，即参考模型和原型模型，其中参考模型用于定义临床信息的组织结构，原型模型用来约束信息的语意，实现了数据的底层存储和语意表达的分隔。在临床场景下，不同类型的医疗数据通过原型模型来表达，基于原型查询语言便可获得原型对应医疗数据记录。基于以上分析，决定可视化组件基于原型进行设计和开发，可视化组件与原型的绑定并通过原型查询接口实现可视化

组件数据源的获取从而实现数据绑定，其中，原型查询接口返回结果采用轻量级的数据交换格式 JSON 来表达，可视化组件通过 JSON 解析实现数据显示。

- 2) 针对扩展模块之间交互能力不足的问题，论文剖析了动态扩展技术，选择采用插件技术来实现集成可视化系统的动态扩展，系统的扩展模块包括集成视图和可视化组件两部分。通过为插件开放通用通信接口的方式，来实现各扩展模块的相互通信，增强系统的交互特性。
- 3) 基于以上系统设计，对系统进行临床实践。基于 openEHR 原型开发可视化组件，并通过原型绑定来实现可视化组件所需数据源的绑定。开发了时间导航视图、就诊导航视图、检验一览视图和检验一览视图等集成视图，且各视图之间及视图与可视化组件（插件）之间可通过通信接口实现灵活的通信。

本论文提出的医疗数据集成可视化系统的动态可扩展框架，具有比较低的耦合性且系统组件可以被很好复用，通过可视化组件与 openEHR 原型的绑定可满足数据访问，实现了基于领域知识的可视化扩展，同时扩展模块之间通过开放通信接口，实现了灵活的通信。该系统已在国内某三甲医院全部门诊进行了临床应用，应用状况良好。

5.2 展望

本论文提出的可动态扩展的医疗数据集成可视化系统，虽然在一定程度上能实现新增医疗数据和满足不同临床场景的集成浏览样式的动态扩展，可以避免系统与底层数据库的高耦合，以及扩展模块之间可以实现灵活的交互，但在工作中仍然存在很多不足，主要表现在如下几个方面：

- 本论文所提出的基于 openEHR 原型的数据绑定方法，通过 XML 配置文件实现 dADL 到 JSON 映射，当原型发生大的结构变化时，很难直接通过更改 XML 配置文件的更改实现重新映射，下一步可以通过 XML+脚本的方式实现映射，XML 文件实现简单的映射关系，脚本实现复杂的映射关系。
- 系统的个性化配置需要用户在视图订制工具中手动进行设置，可以进一步通过对用户使用系统的行为进行跟踪分析，实现视图自动化订阅或推荐功能。
- 系统只是单纯的对数据进行展示，下一步可以引入一些智能化应用，在展示数据的同时能将数据背后隐藏的知识以可视化的方式展现给医生。

参考文献

- [1] AN J, LU X, DUAN H. Integrated visualization of multi-modal electronic health record data; proceedings of the Bioinformatics and Biomedical Engineering, 2008 ICBBE 2008 The 2nd International Conference on, F, 2008 [C]. IEEE.
- [2] PLAISANT C, MILASH B, ROSE A, et al. LifeLines: visualizing personal histories; proceedings of the Proceedings of the SIGCHI conference on Human factors in computing systems, F, 1996 [C]. ACM.
- [3] POWSNER S M, TUFTE E R. Graphical summary of patient status [J]. The Lancet, 1994, 344(8919): 386-9.
- [4] PLAISANT C, ROSE A. Exploring LifeLines to visualize patient records [J]. 1998,
- [5] BUI A A, TAIRA R K, CHURCHILL B, et al. Integrated visualization of problemcentric urologic patient records [J]. Annals of the New York Academy of Sciences, 2002, 980(1): 267-77.
- [6] BUI A A, ABERLE D R, KANGARLOO H. TimeLine: visualizing integrated patient records [J]. Information Technology in Biomedicine, IEEE Transactions on, 2007, 11(4): 462-73.
- [7] 陈湖山. 可动态配置的电子病历数据集成视图研究与开发 [D]; 浙江大学, 2012.
- [8] IBM Research Unveils 3D Avatar to Help Doctors Visualize Patient Records and Improve Care [M].
- [9] 竺银瑶. 基于行为关系的电子病历数据集成可视化研究[D]. 浙江大学生物医学工程与仪器科学学院 浙江大学, 2010.
- 浙江大学, 2010.
- [10] 电子医疗助理, Last Cited on 2016.01.04. <https://www.modmed.com/>
- [11] ZEIGER R. Google Body becomes Zygote Body; built on open source 3D viewer [J]. Google Open Source blog, 2012,
- [12] BEALE T, HEARD S. openEHR Architecture: Architecture Overview. London: 2008 [M].
- [13] 罗智勇, 罗娟, 赖德军. 针对 WFS 服务的 JSON 数据传输方案研究 [J]. 地理与地理信息科学, 2012, 28(1): 111-2.
- [14] CROCKFORD D. Introducing json [J]. Available: json org, 2016,
- [15] 李浩, 王恂. 用 JSON 改进 AJAX 数据传输 [J]. 中国高新技术企业, 2008, 16): 123-.

- [16]独立 JSON 序列化, Last cited on 2016.01.03.
[https://msdn.microsoft.com/zh-cn/library/bb412170\(v=vs.110\).aspx](https://msdn.microsoft.com/zh-cn/library/bb412170(v=vs.110).aspx)
- [17]KALRA D, BEALE T, HEARD S. The openEHR foundation [J]. Studies in health technology and informatics, 2005, 115(1):53-73.
- [18]曾蕾, 姚志洪, 刘雷. 双模型健康档案标准 openEHR [J]. 中国医疗设备 ISTIC, 2010, 25(3):
- [19]王利, 闵令通, 吕旭东, et al. 基于 openEHR 的原型关系映射方法 [J]. 中国生物医学工程学报, 2014, 31(4): 432-7.
- [20]李珍珍. 基于 openEHR 的电子病历系统开发方法研究与实践 [D]; 浙江大学, 2008.
- [21]Beale T. Archetype Query Language (AQL)(Ocean)[J]. OpenEHR, Adelaide, viewed September, 2008, 25: 2009.
- [22]石怡. WPF 使用 XAML 实现对 SQL Server 数据绑定的方法 [J]. 电脑开发与应用, 2011, 24(10): 70-1.
- [23]数据绑定概述, Last Cited on 2016.01.03
[https://msdn.microsoft.com/zh-cn/library/ms752347\(v=vs.110\).aspx](https://msdn.microsoft.com/zh-cn/library/ms752347(v=vs.110).aspx)
- [24]王利, 闵令通, 吕旭东, et al. 高度可扩展的 CDR 构建技术研究 [M]. 2014 中华医院信息网络大会论文集. 大连. 2014: 1-4.
- [25]陈方明, 陈奇. 基于插件思想的可重用软件设计与实现 [J]. 计算机工程与设计, 2005, 26(1): 172-6.
- [26]蔡勇, 刘红海, 郑小军, et al. 基于 Windows CE 平台的 COM 组件及其在串口通信中的应用 [J]. 计算机工程与设计, 2005, 25(12): 2363-5.
- [27]潘爱民. COM 原理与应用 [M]. 清华大学出版社, 1999.
- [28]杨秀章. COM 技术内幕 [M]. 北京: 清华大学出版社. 1999.
- [29]戴宗友, 汪涛. 浅析 ActiveX, OLE 和 COM [J]. 电脑知识与技术, 2001, 9(4):2-3.
- [30]元数据和自描述组件, Last Cited on 2016.01.03.
[https://msdn.microsoft.com/zh-cn/library/xcd8txaw\(v=vs.110\).aspx](https://msdn.microsoft.com/zh-cn/library/xcd8txaw(v=vs.110).aspx)
- [31]张强, 周荣辉. COM 和 .NET 构件模型 [J]. 教育信息化, 2006, (5): 36-8.
- [32]Managed Extensibility Framework (MEF) , Last Cited on 2016.01.03.
[https://msdn.microsoft.com/en-us/library/dd460648\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd460648(v=vs.110).aspx)
- [33]李凤桐, 杨艳峰, 赵景超. 基于 MEF 技术的 “插件式” 应用系统解决方案 [J]. 电脑编程技巧与维护, 2015(4): 93-4.

[34] 外接程序和扩展性, Last Cited on 2016.01.03.

[https://msdn.microsoft.com/library/bb384200\(v=vs.100\).aspx](https://msdn.microsoft.com/library/bb384200(v=vs.100).aspx)

[35] 管线开发要求, Last Cited on 2016.01.03.

[https://msdn.microsoft.com/zh-cn/library/bb384240\(v=vs.110\).aspx](https://msdn.microsoft.com/zh-cn/library/bb384240(v=vs.110).aspx)

[36] 王亚, 宋铭利. Windows 消息机制研究 [J]. 现代计算机: 下半月版, 2008, 2): 70-1.

[37] 杨亮, 阮晓星. WINDOWS 消息驱动机制中的核心技术分析 [J]. 计算机应用研究, 1997, 14(5): 12-4.

[38] 吴良波. ISV 插件平台系统设计与实现 [D] [D]; 浙江大学, 2010.

[39] 薛允民. 浅谈医疗检查检验结果的质量控制与互认 [M]. 2007 中国计量论坛论文集. 乌鲁木齐. 2007: 92-6.

作者简介

作者在攻读硕士期间参与的项目：

- 1) 2013 年 8 月至 2014 年 12 月参与国内某三甲医院数字化医院建设项目，负责医学影像信息系统的研发工作。
- 2) 2014 年 2 月至 2015 年 8 月参与国内某三甲医院全院信息系统集成融合项目，并参与全院高端电子病历系统的研发。

附录一

1) 检查影像可视化组件 manifest.json 文件内容

```
{
  "Version": "1.0.0.3",
  "Name": "医学影像可视化组件",
  "PluginFile": "DICOMViewer.dll",
  "Dependency": [
    "Interop.IMagicLib.dll",
    "AxInterop.IMagicLib.dll",
    "IHDFtpClient.dll",
    "Newtonsoft.Json.dll"
  ],
  " PrototypeBinding ":[
    "openEHR-EHR-OBSERVATION.imaging_exam_image_series.v1"
  ]
}
```

2) 心电波形可视化组件 manifest.json 文件内容

```
{
  "Version": "1.0.0.3",
  "Name": "心电波形可视化组件",
  "PluginFile": "ECGViewer.dll",
  "Dependency": [
    "EcgFrame.dll",
    "Interop.ECGViewLib.dll",

```



```
        "AxInterop.ECGViewLib.dll",  
        "IHDFtpClient.dll",  
        "Newtonsoft.Json.dll"  
    ],  
    "Prototype": [  
        "openEHR-EHR-OBSERVATION.imaging_exam_image_series.v1"  
    ]  
}
```

3) 检查报告可视化组件 manifest.json 文件内容

```
{  
    "Version": "1.0.0.3",  
    "Name": "检查报告可视化组件",  
    "PluginFile": "DocumentViewer.dll",  
    "Dependency": [  
        "AxInterop.AcroPDFLib.dll",  
        "Interop.AcroPDFLib.dll",  
        "Newtonsoft.Json.dll",  
        "IHDFtpClient.dll",  
        "ICSharpCode.SharpZipLib.dll"  
    ],  
    "PrototypeBinding": [  
        "openEHR-EHR-OBSERVATION.imaging_exam_report.v1"  
    ]  
}
```

4) 检验数据可视化组件 manifest.json 文件内容

```
{
  "Version": "1.0.0.3",
  "Name": "检验数据可视化组件",
  "PluginFile": " LabTestViewLib.dll",
  "Dependency": [
    "Newtonsoft.Json.dll"
  ],
  " PrototypeBinding":[
    "openEHR-EHR-OBSERVATION.lab_test.v1",
    " openEHR-EHR-ACTION.Lab_test.v1 "
  ]
}
```

5) 手术数据可视化组件 manifest.json 文件内容

```
{
  "Version": "1.0.0.3",
  "Name": "手术数据可视化组件",
  "PluginFile": " LabTestViewLib.dll",
  "Dependency": [
    "Newtonsoft.Json.dll"
  ],
  " PrototypeBinding ":[
    " openEHR-EHR-INSTRUCTION.operation_arrangement.v1",
    " openEHR-EHR-ACTION.Operation.v1.adl "
  ]
}
```

6) 处方数据可视化组件 manifest.json 文件内容

```
{
```

```
"Version": "1.0.0.3",  
"Name": "处方可视化组件",  
"PluginFile": " DocumentViewer.dll",  
"Dependency": [  
    " AxInterop.AcroPDFLib.dll",  
    " Interop.AcroPDFLib.dll",  
    " Newtonsoft.Json.dll",  
    "IHDFtpClient.dll",  
    " ICSharpCode.SharpZipLib.dll"  
],  
" PrototypeBinding ":[  
    "openEHR-EHR-INSTRUCTION. prescription. v1",  
    " openEHR-EHR-ACTION. Prescription. v1 "  
]  
}
```

7) 医嘱数据可视组件 manifest.json 文件内容

```
{  
    "Version": "1.0.0.3",  
    "Name": "医嘱数据可视化组件",  
    "PluginFile": " OperationViewLib.dll",  
    "Dependency": [  
        " Newtonsoft.Json.dll"  
    ],  
    " PrototypeBinding ":[  
        "openEHR-EHR-INSTRUCTION. order. v1",  
        "openEHR-EHR-ACTION. order. v1",  
    ]  
}
```

```
        "openEHR-EHR-ACTION. order_excute. v1"
    ]
}
```

8) 病历数据可视组件 manifest.json 文件内容

```
{
  "Version": "1.0.0.3",
  "Name": "病历数据可视化组件",
  "PluginFile": "EMR.dll",
  " PrototypeBinding ": [
    " Newtonsoft.Json.dll",
    "Interop.MRWRITERLib.dll"
    "AxInterop.MRWRITERLib.dll"
  ],
  " PrototypeBinding ":[
    "openEHR-EHR-OBSERVATION. emr_document. v1"
  ]
}
```