**设计报告**

曼联是冠军

1. **题目分析**

题目是要实现一个类似Windows注册表编辑器的管理系统。注册表编辑器中的数据分为两类，一类是key，类似于文件；另一类是value和data的键值对（其中每个value有特定的类型），每个键值对都属于某一个key，类似于key的属性。因此注册表相当于文件和文件属性的集合。

一般的文件系统中，目录也是一种文件，只是拥有特殊类型的文件；而不同于文件系统，注册表中的“文件”就是、也只能是“目录文件”。换句话说，不区分文件和目录，所有的文件都是目录。

抽象出来，整个注册表就是一棵树，拥有单个隐藏的根节点。每个key的位置标明了其在树中的位置。所有value和data的键值对是树中节点的附加数据。根据这样的思路即可完成题目。

1. **技术路线**
   1. **数据储存方式**

给出的测试用例是Windows注册表导出后的文本文件。如果采用这样的文本文件保存数据，缺陷是非常明显的。保存方式基本上是这样的两种选择：第一种是每次操作都遍历整棵树并更新整个文件——实现容易，但是效率太低；第二种是记录每条数据在文本文件中的位置，每次操作都到文本文件中寻找对应的位置并做出部分修改——实现过于复杂，并且效率提高并不明显。

给出的测试用例是Windows注册表导出后的文件。实际上Windows注册表并不是以文本文件管理的，而是操作系统内建了一个层级数据库。数据库的文件以二进制的格式存放在%WINDOWS%\SYSTEM32\CONFIG和%USER%\NTUSER.DAT中。

因此，为了持久化注册表数据，并且以方便快速的形式读写，应该采用数据库来储存。

为了系统的稳定性、效率和实现的方便，应采用成熟的数据库。虽然注册表是层级结构，但是即使在可以储存层级结构的非关系型数据库中，一般对数据层级数目是有限制的，并且储存过深的层级数据会显著降低效率，因此考虑所有key平级储存的单层结构，每个key储存一个“路径”数据，标明其位置。key的value-data键值对属性附加至key上储存。这里参考了MongoDB官方手册中介绍的储存方式：<http://docs.mongodb.org/manual/tutorial/model-tree-structures-with-materialized-paths/>。

这样做的优势有：1.储存方便快速，可扩展性强，即使注册表文件非常大，也只需要读取一次，储存起来也毫无压力；2.读取key的属性时无需遍历树结构，根据路径即可直接匹配到；3.由于所有key在同一层级，在任意位置添加新的key、或是增加、修改、删除某个key的属性非常直接方便快速。缺点是修改某个key的名字需要遍历其所有子节点并全部修改，因此在修改上层key的名字时可能速度较慢。

不过分析实际的场景可以发现，“修改上层key名”的操作是非常罕见的，而添加key（安装软件）和添加、修改、删除某个key属性（配置软件）的操作特别频繁，因此在这样的使用场景下，应该采用这样的单层数据结构的储存方式。

这样的数据库多种多样，总体来说分为关系型数据库和非关系型数据库。如果使用关系型数据库，需要建立至少两张表：一张表用于储存所有的key信息，包括名字和路径；另一张表用于储存所有的属性，并通过key的ID与之关联。使用非关系型数据库则是将每个key存为一个文档，key的路径和属性作为文档的属性和文档绑定储存。

使用非关系型数据库更为直接地将value、data键值对与key绑定在一起，并且导出的时候无需像关系型数据库那样使用大量速度很慢的联合查询，因此我们最终决定采用非关系型数据库。实际实现中我们采用了目前行业内最为成熟的非关系型数据库MongoDB作为数据储存方式。

* 1. **数据读取方式**

如果在内存中建立整棵注册表的树结构，好处是建立后就可以很快地读到每个节点和其属性。然而注册表导出后一般都超过了100MB，虽然目前计算机的内存单位是GB，但是一方面树结构本身也要占用附加的空间；另一方面、也是更重要的是，管理数据库的时候长期在内存中维护一棵超过100MB的数据结构、并且很少会操作所有的节点，这显然是一个低效的方法。而且这样的选择在每次打开该系统的时候都需要重新建立树，初始化速度过慢。

因此应该采取部分读入的方式，先读入最高层的key，用户在需要打开下级key的时候再去读取相应的key。

* 1. **GUI的选择**

目前常用的GUI库有MFC、GTK、wxWidgets和Qt。MFC相对比较古老，多语言支持能力一般，不能跨平台，并且由于我们的电脑上没有Windows操作系统，也没有足够的钱购买VS，因此忽略；GTK由于纯C语言复杂的实现和跨平台特性问题也不予采用；wxWidgets和MFC颇为相像，而且在对象封装和功能上比MFC强很多，只是在多语言支持上稍有问题；Qt在各方面都非常强大，跨平台能力和多语言支持几乎完美，并且文档和库也非常多，只是其信号/槽机制需要通过自身的预编译器处理，编译的时候需要使用Qt提供的qmake而不能单独使用C++的编译器。

除了要本地操作以外，题目还要求能够实现远程管理。如果使用上述的GUI库，虽然网络通信在各个GUI库中都已有成熟的方案，但这势必还是会增加不少的工作量。

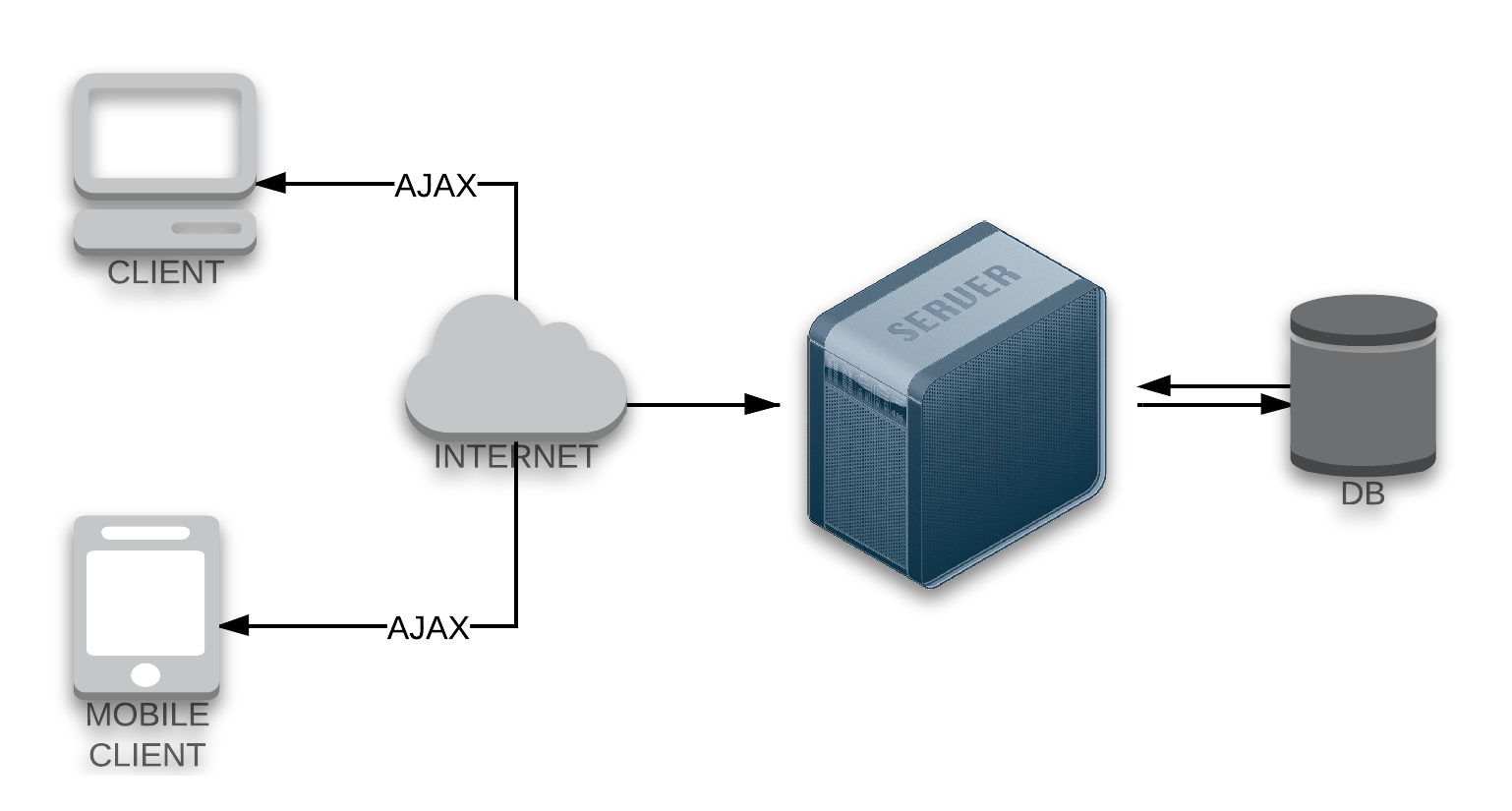
于是，考虑到远程管理的问题，再考虑到各个GUI库的优缺点，我们换了一种思路，可以满足完全的跨平台和多语言支持能力、丰富的库和资源、方便简单的开发方式、以及原生的远程管理能力——我们决定采用浏览器-服务器的模型来完成。也就是搭建一个服务器，将整个管理系统建立在Web页面上。前端的图形操作开发采用浏览器可以直接解释的HTML、CSS和JavaScript来完成，相应的库和资源非常多，这里我们采用了Twitter推出的Bootstrap作为基本的界面框架，采用jsTree作为树形结构的显示库。这样运行之后既可以在本地通过浏览器访问本地地址来管理，也可以在网络上通过对应的IP（或者域名）来访问。

* 1. **开发语言的选择**

开发的过程分为了浏览器（前端）和服务器（后端）。由于浏览器能解析的语言只有JavaScript，因此我们直接采用JavaScript，配合jQuery库进行前端开发。

后端语言的选择多种多样，但是由于最终要提交可执行文件，如果使用Apache或者Nginx作为服务器，使用脚本语言开发，最后很难提交。用后台语言自身搭建服务器，并且可以生成可执行文件，最终我们选择了Go语言作为后台开发语言。Go语言是静态强类型语言，方便生成可执行文件，跨平台能力强；Go拥有开发良好的MongoDB库，其自身搭建的服务器性能也很好，可以支持大量并发访问；Go在语言层对多线程的支持非常好，在本题目的一些耗时的操作上（如修改key的名字）可以采用多线程，可以大大提高效率。

1. **总体方案**

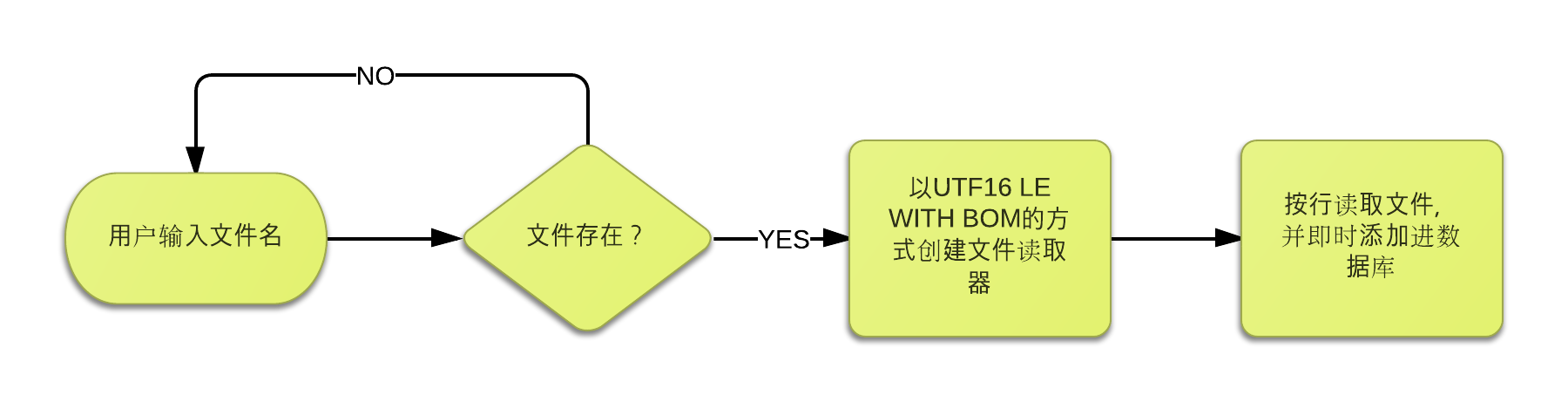


页面与服务器采用Ajax方式进行通信，整个过程没有页面跳转。

服务器与数据库建立连接进行查询、添加、修改和删除的操作。

1. **具体实现**
   1. **文件导入**

下图为导入的流程：



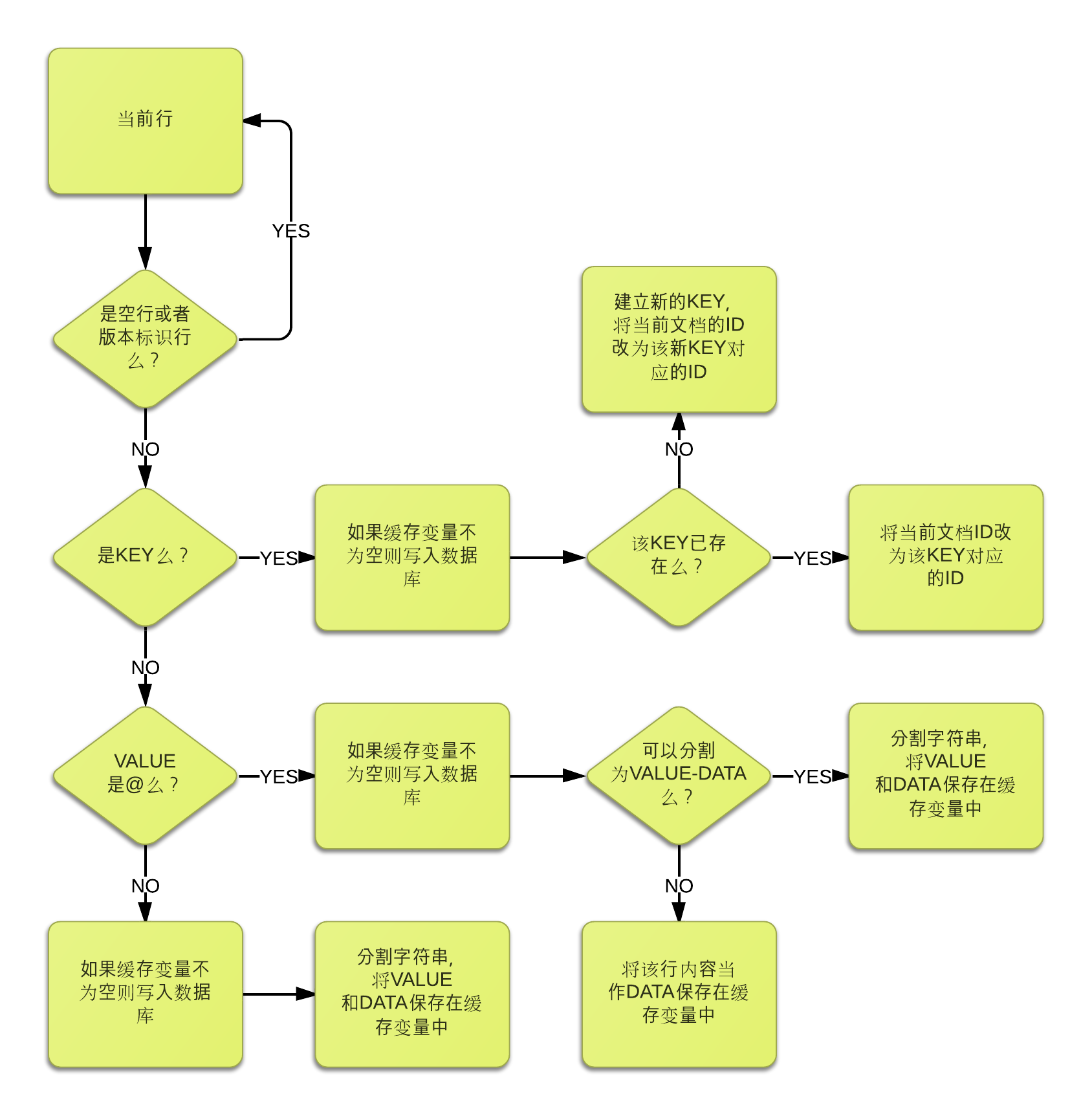
接下来分析文件读入的流程。这里我们查看了实际的数据，发现有大量复杂的规则在测试用例中没有体现。比如实际的注册表导出文件中，数据是可以换行的，并且二进制数据换行的时候行尾会有反斜杠，其他格式的数据换行时行尾什么都没有；每个key会有一个默认的value，名字是@；key的名字除了反斜杠外可以是任何字符，而value可以是任何字符。

这些复杂的情况造成的最直接的问题就是如何分割value-data键值对。直接以等号分割是一定不行的，因为value里也可以有等号；value和data的两边有可能有双引号，有可能没有，因此也不能用”=”这样的字符串来分割；更复杂的地方是，value两边有引号，value里面也有引号和等号（里面的引号前要用反斜杠转义）。综合这些情况来考虑，最终我们写了一个正则表达式：[^\\](\\\\)\*"=，用于匹配value两边有引号的分割。基本原理是，寻找”=字符串的位置，但是为了避免"\"="的情况，要求"=字符串前面不能有反斜杠；但是如果是"\\"=这样的字符串应该依然是合法的。因此我们限定"=前面没有或者只能有偶数个反斜杠。对于value两边没有引号的情况，我们发现只有@这一种情况，因此单独处理即可。

接下来是换行的问题。由于换行的地方不一定有标识，因此采取的方法是每当出现key或者value-data键值对的第一行，就把之前缓存的value-data键值对存入数据库，否则将当前读取的value-data键值对放入缓存变量中。

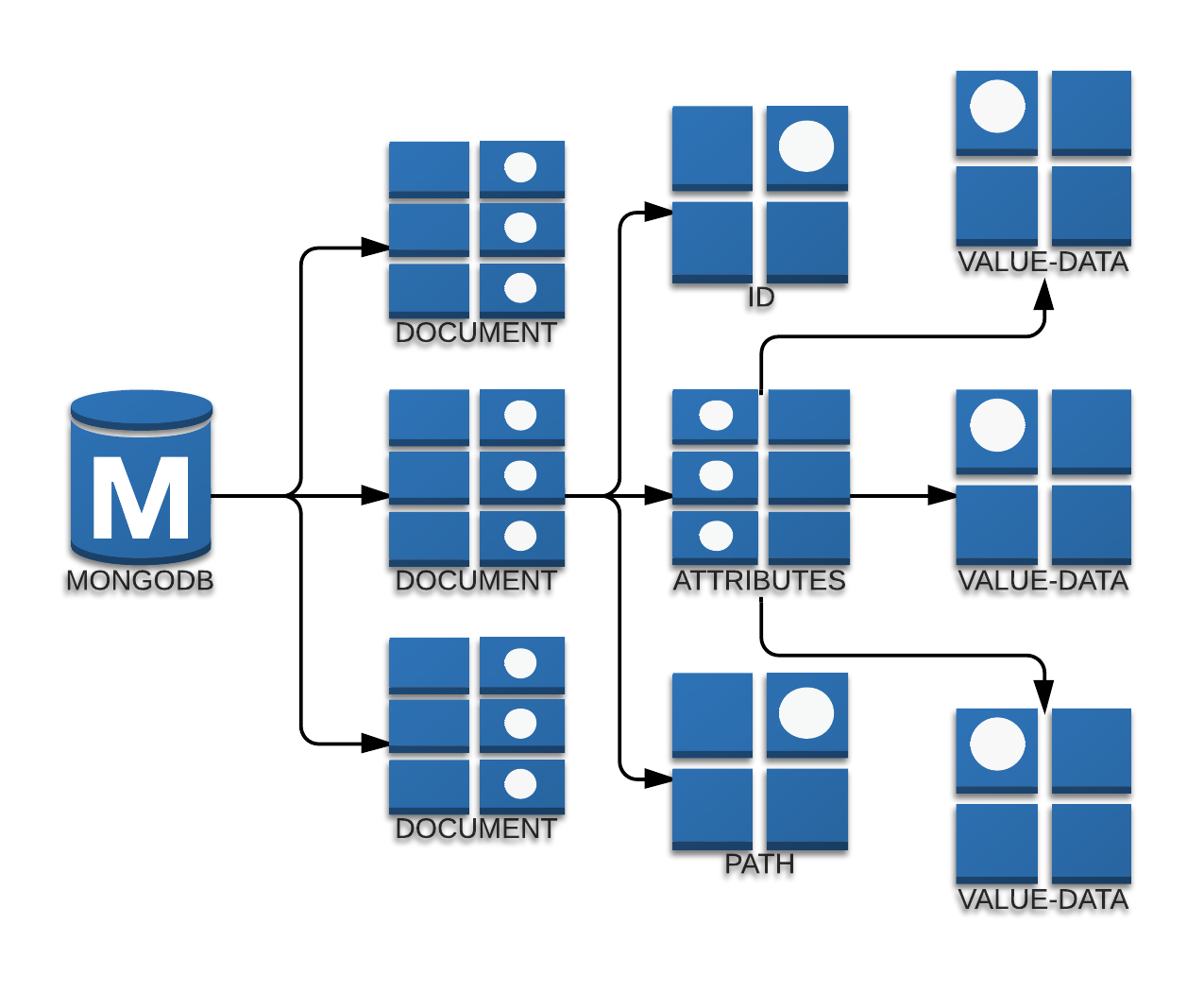
另外，实际的注册表的开头会有一行文本标明注册表编辑器的版本号，也应该检测去除。

下图为实现流程：



将数据写入数据库之前，先判断数据的类型：带双引号的为REG\_SZ，开头为dword:的为REG\_DWORD，开头为hex:的为REG\_BINARY，开头为hex(#):的也有对应的好几种数据格式，根据微软官方的文档判断类型之后将数据的类型前缀去掉，同时将类型和数据存入对应的key下。

下图为数据储存的方式：



具体的数据结构为：{"\_id": ID, "path": PATH, "attr": {value1: {"type": type1, "data": data1}, value2: {"type": type2, "data": data2}, ...}, "text": text}

* 1. **文件导出**

遍历数据库，每个文档先输出key，再输出value-data键值对，同时根据储存的类型，还原数据前缀。

* 1. **key操作**

添加key时只需新建一个文档，将key的名字和添加的位置路径存入即可。

修改key时需要在所有文档中查找路径前缀为该key路径的所有文档，并循环修改。

删除key时和修改步骤类似，只是无需循环修改，而是直接删除。

key的名字和路径由前端的jsTree库提供。

* 1. **value和data的操作**

添加value-data键值对时需要先判断该value是否已存在于该key中，不存在的话即可在对应的key下添加一条键值对。

修改value-data键值对时需要先判断value是否被更改，没被更改的话就直接完成data的修改；如果被更改的话需要判断该value是否已存在与该key中，不存在的话即可完成value和data的修改。

删除value-data键值对时只需在该key下找到对应的value即可删除value-data键值对。

* 1. **前端和后台的数据统一**

在载入页面的时候前端会自动读取最顶层的key，并保存在一个JavaScript对象中。之后每次操作（展开、添加、修改、删除key，和添加、修改、删除value-data键值对）时都在该对象中对应修改，同时向后台发送操作的请求，以保持前端页面和后台的统一。

在操作key的时候jsTree已经实现了动态样式的更改。value-data键值对需要手动进行动态更改，即添加一条数据之后在表格中添加一行、修改一条数据后修改表格中的数据，以及删除一条数据后删除表格中的一行。

1. **使用指南**
   1. **页面结构**

顶部为操作导航栏，包括导入、导出和删除整个注册表的功能。最右侧的链接是本文档。

下面分为左右两侧。左侧为key的树结构，右侧为当前选中的key对应的value-data键值对，还有data的类型。

* 1. **操作说明**

运行可执行文件，默认的访问地址是http://localhost:2048。另外，我们在实际的服务器上也搭建了该系统，访问地址是<http://winreg.melkor.me>。

点击顶栏的Import按钮，输入要导入的注册表文件名（将注册表文件放在执行文件同级的目录下），点击Import。导入结束后页面会自动刷新，此时可以看到注册表最顶级的key。

左侧的树形结构中，点击key左侧的三角形可以展开，点击key在右侧会显示该key对应的value-data键值对；右击key可以进行如下操作：添加（Create），修改（Rename）和删除（Delete），剪切（Cut）、复制（Copy）和粘贴（Paste）暂不可用。

右侧的表格中，点击每行最后的修改按钮会弹出修改对话框，可以进行value和data的修改（DWORD类型和BINARY类型只能输入16进制数）；点击每行最后的删除按钮可以删除改value-data键值对；点击最后一行的添加按钮会弹出添加对话框，可以进行value-data键值对的添加和类型选择。

点击顶栏的Output按钮，输入导出的注册表文件名，系统将现在的注册表导出，文件的位置与可执行文件在同级目录下。

点击顶栏的Drop按钮可以清空当前的注册表。

在顶栏的搜索框中可以搜索key的名字。

点击顶栏的Doc可以查看本文档。