存储模块设计文档

***2016013258** 王泽宇*

***2016013270** 李帅*

***2016013276** 纳鑫*

1.引言

1.1编写目的

本文档是我们小组编写的数据库的存储模块的设计文档，目的是阐述清楚存储模块的实现方式、功能，方便检查进度，以及之后项目编写时查找函数接口使用。

1.2背景

我们的项目名称是`NaiveDB`，开发者共三人

- 王泽宇，邮箱：ycdfwzy@outlook.com
- 李帅，邮箱：lishuai16THU@163.com
- 纳鑫，邮箱：naxinlegend@outlook.com

潜在用户：需要使用一些简单数据存储的开发者。

2.总体设计

2.1需求规定

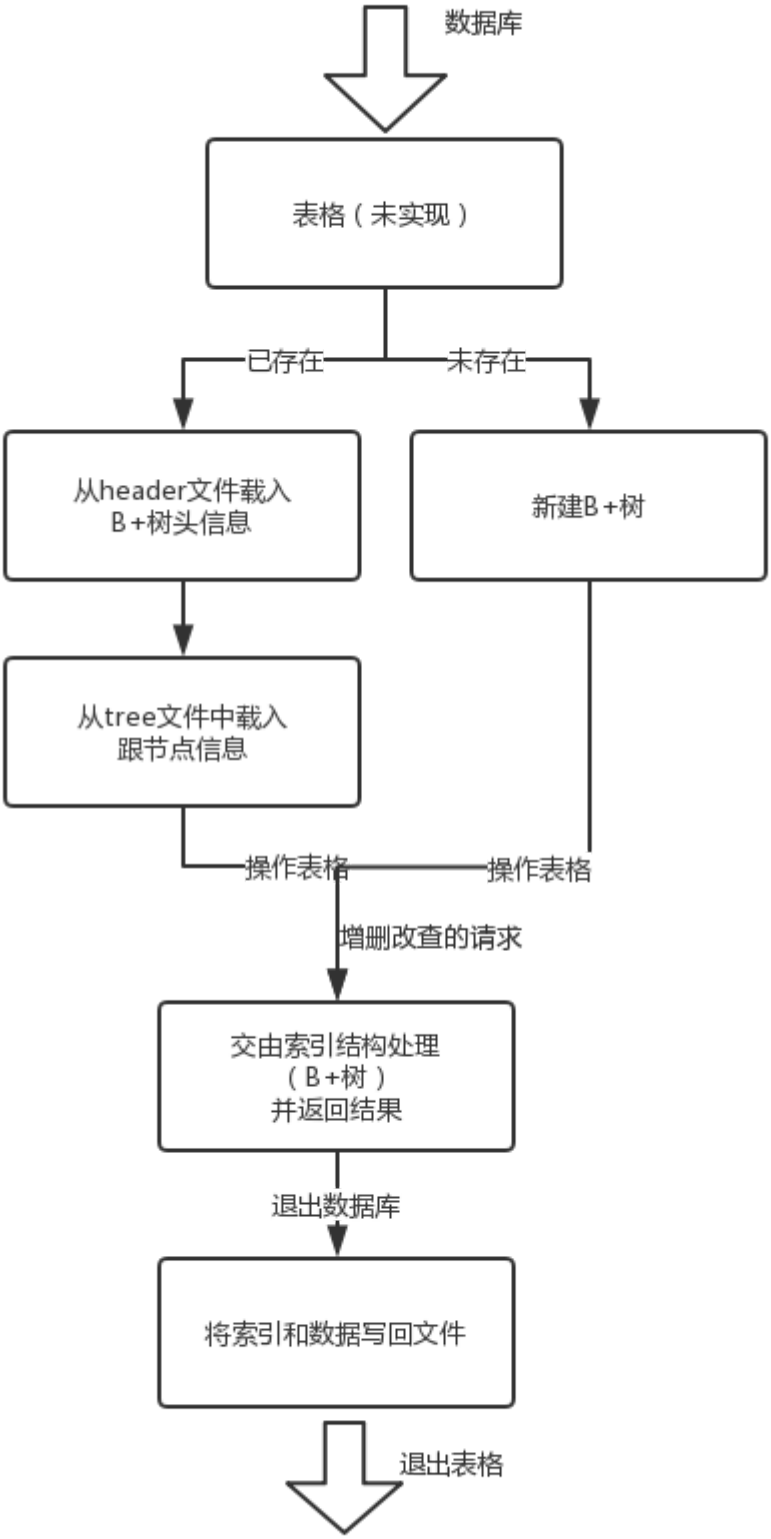
系统要求能够将包含五种数据类型（Int，Long，Float，Double，String）的记录可持久化存储到磁盘上（自定义存储格式），并且能够对这些记录进行增删改查四种基本操作。

2.2 运行环境

项目使用Java语言开发，JDK版本：1.8.0_201

2.3基本设计概念和处理流程

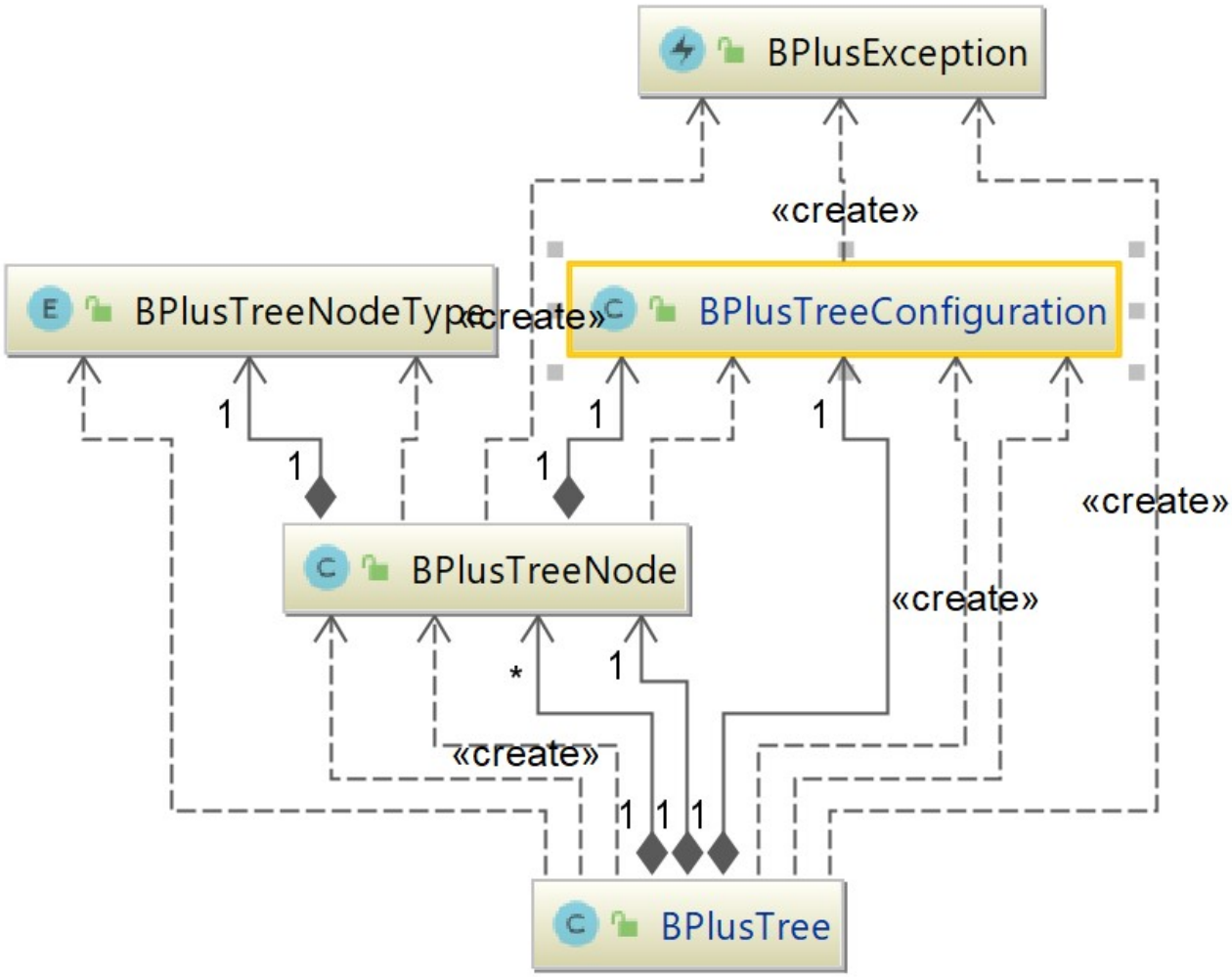
基本流程如下图所示



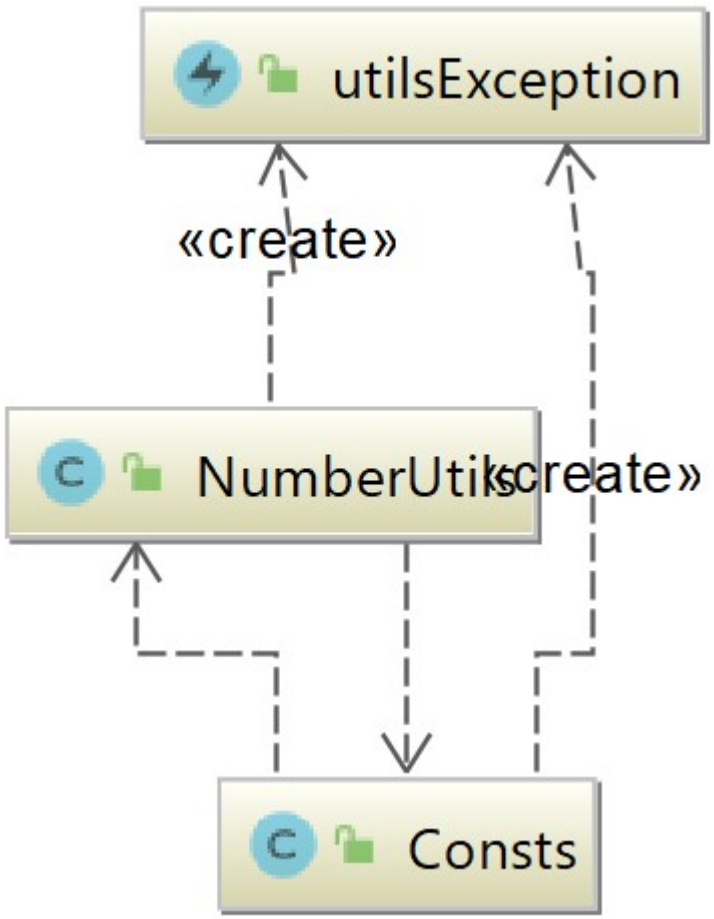
2.4结构

下面是B+树的文件结构

- BPlusTree中会存储根节点BPlusTreeNode（一对一关系），还会存储若干个缓存的节点（一对多关系），还会有一个变量记录配置信息BPlusConfiguration（一对一关系）。
- BPlusTreeNode中也会存储该树的配置信息（一对一关系），还会存储节点的类型BPlusTreeNodeType（一对一关系）。
- 最终所有类都会将异常抛出BPlusException中。



下面是utils包的文件结构。由于utils的功能是提供一些常用的常量/函数，所以相互之间的关系没有B+树的复杂，只是如果其中的类遇到了异常，都会将异常抛出给`utilsException`。



2.5尚未解决的问题

为了方便测试，目前在世把索引和数据都写在了B+树的类中，我们决定在之后的开发中将数据部分从B+树的部分分离出来。

3.接口设计

3.1 BPlusTree包

这个包对外提供两个类：`BPlusTree`类和`BPlusTreeConfiguration`类

3.1.1BPlusTree类

- 提供两种构造函数，`BPlusTree(BPlusTreeConfiguration config)`使用一个`config`构造一个全新的B+树，`BPlusTree(String filename)`从文件名为`filename`的存储文件载入B+树，
- 插入`insert(LinkedList values)`，将记录`values`插入树中，其中`values`的元素存储顺序应与config中的类型顺序一致，也就是说第一个元素应是主键。
- 删除`delete(Object value)`，将主键为`value`的记录删除
- 修改`update(LinkedList values)`，将主键为`values[0]`的记录修改为`values`
- 查找`search(Object value)`，查找主键为`value`的记录
- 关闭`close()`，这个函数需要在函数析构前显示调用，以将内存中的修改全部写回到文件中去。

3.1.2BPlusTreeConfiguration类

这个类存储了B+树的基本信息，包括存储在文件中的文件名`filename`，每个page的大小`pageSize`（单位Byte），每一个属性的类型`columnType`（第一个属性为主键），B+树的度数`treeDegree`（根据`pageSize`和`columnType`算得）。

构造函数需要提供`pageSize`，`filename`，`keyType`，`columnType`，其中`pageSize`是可选参数，默认值为4096。需要注意的是，如果`pageSize`太小，导致算出的树度数不满3，程序会抛出异常。或者`keyType`和`columnType[0]`不同，程序也会抛出异常。

3.2 utils包

该程序包中包含了一些常用的工具

3.2.1 Consts类

存储了一些必要的常量，包括各种数据类型的在文件中存储的默认长度等。 支持的数据类型包括：

- Int：整型，范围$-2^{31} \sim 2^{31}-1$
- Long：长整型，范围$-2^{-63} \sim 2^{63}-1$
- Float：单精度浮点数
- Double：双精度浮点数
- String×××：定长字符串，表示`×××`字符串长度，如果省略，默认为256。

有一个对外的函数`Type2Size(String)`，接受表示类型的字符串，返回字符串的长度。

3.2.2 NumberUtils类

仅对外提供一些使用的函数，包括

- `isPureInteger(String)`,`isPositiveInteger(String)`,`isNegativeInteger`,`isInteger`,`isFloat`：判断字符串是否是纯整数（不带正负号），正数（可带正号），负数，整数，小数。
- `parseInt(str,s,len,flag=false)`：将`str[s~s+len]`解析为整型数返回，如果flag为`true`，则允许结果为`null`，否则解析到`null`会抛出异常。类似的函数还有`parseLong`，`parseFloat`，`parseDouble`，`parseString`。
- `readInt(BufferedInputStream)`,`writeInt(BufferedOutputStream, int)`：从文件中读入整型数，或者将整数存储到文件中去。类似的函数有`readLong`,`writeLong`,`readFloat`,`writeFloat`,`readDouble`,`writeDouble`,`readString`,`writeString`,关于String的读写可以提供一个长度参数，指明String的长度，否则长度为String的默认长度。
- `fromBytes(list,str,pos,type,flag)`：从`str`的`pos`位置开始，解析出`type`类型的数据，并加到`list`里。如果flag为`true`，则允许解析结果为`null`，否则解析到`null`会抛出异常。
- `toBytes(byte[] bytes,pos,Object value,type)`：将`value`按`type`类型转化为`byte[]`，并存入bytes的`pos`位置。

4.系统数据结构设计

4.1 可持久化存储文件格式

B+树会被存储到两种文件中去

- `*.header`文件存储的是整棵树的头信息。

> pageSize|columnCnt|[columnTypes]|rootPageIndex|first_leaf|last_leaf|blankPageCnt|[blankPageIndex]

1. `pageSize`即叶节点在磁盘上存储的页大小
2. `columnCnt`表示属性的数量
3. `[columnTypes]`一共长度为`columnCnt`的字符数组，存储了每一个属性的类型，第一个属性为主键
4. `rootPageIndex`根节点在所在页的编号（从0开始）
5. `first_leaf`第一个叶子所在页的编号
6. `last_leaf`最后一个叶子所在页的编号
7. `blankPageCnt`空页的数量（用于删除节点后的碎片整理）
8. `[blankPageIndex]`长度为`blankPageCnt`的数组，存储了空页的编号。

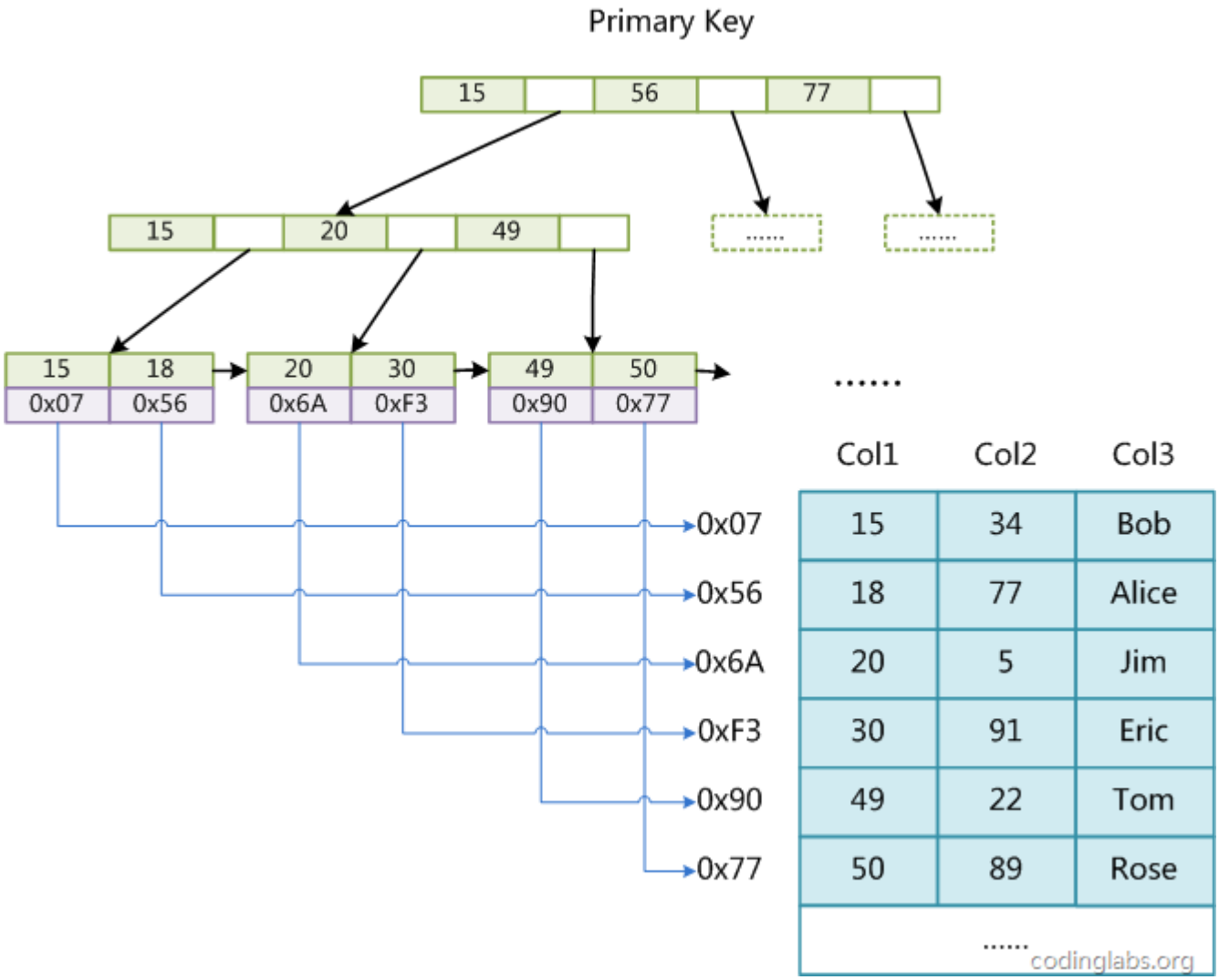
- `*.tree`文件存储的是B+树的节点信息。一共有若干页，一个节点存在一页（page）中，节点的格式如下：

> node_type|parent_ptr(|next_page|prev_page)|[(key, ptr)]

1. `node_type`表示节点的类型：叶子节点、根节点、内部节点。
2. `parent_ptr`父节点所在页的编号。
3. `next_page`，`prev_page`，这两个是只有叶子节点会有的信息，表示上一个叶子或者下一个叶子所在页的编号。
4. `[(key, ptr)]`，存储了键值和`ptr`对的数组，如果该节点是非叶子节点，`ptr`指的是子节点所在也得编号，如果是叶子节点，`ptr`指的是该条记录在数据文件中的行号。

数据会被存储到`.data`文件中去，`.data`文件以`rowSize`作为一条记录的大小（根据`columnType`算出），像数组一样将记录排列下去，第i条记录存储在文件的$i \times rowSize \sim (i+1) \times rowSize$处。

4.2 B+树



我们采用B+树作为索引结构，树的结构如上图所示（图片来源）。增删改查等操作和一般的B+树相同，在此不再赘述。

树结构和数据都会被存储到记录中去，这样每次访问或者修改节点/数据都会造成磁盘的读写。为了缓解这样的状况，我们实现了一个简单的缓存机制：在内存中开辟不超过某一固定值（默认值是1MB，可在`src/utils/Consts.java`中修改）的空间用于缓存一些节点，如果缓存的节点大小超出这一固定值，则随机从缓存中剔除节点。数据的缓存机制和索引的缓存类似。

此外，为了处理删除的节点在文件的位置无法利用的问题（磁盘碎片整理），将被删除空出来的页编号统一存到一个pool中，并将这个pool存到`*.header`文件，保证每次新增的节点都能找到最合适的写入位置（pool为空则写到文件最后，非空则从pool中找一个空位置写入）。

5.系统出错处理设计

对于两个包`BPlusTree`和`utils`，我们都实现了异常类：`BPlusException`和`utilsException`，用于处理在本包中遇到异常，并给出提示信息。例如如果插入的主键已经存在，那么抛出`BPlusException`，如果更新的主键无法找到，那么抛出`BPlusException`，如果发现`parseInt`中发现字符串不可解析为一个数字，则抛出异常.....

所有抛出异常的地方在此不一一列出，详情可以在源码中看到。