# 磁盘分区

文件系统磁盘分区情况如下所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 超级块区 | 用户信息区 | 回收站表 | I节点区 | 数据区 |

超级块：0~2块，大小为3KB

用户信息：3~9块，大小为7KB

回收站map表：10~99块，大小为90KB

i节点：100~3999块，大小为3899KB(3.9MB)

数据区：4000~4294967295块，大小为4092MB(3.996GB)

# 数据结构

超级块

public class SuperBlock

{

//uint.MAX = 1024\*1024\*1024\*4 = 4,294,967,295

public const uint BLOCK\_SIZE = 1024; //块大小

public const uint BLOCK\_SUM\_NUM = 1024 \* 1024 \* 4;//块的总数量，4GB

public const uint iNODE\_SUM\_NUM = 1024 \* 50; //i节点的总数量

public const uint BLOCK\_IN\_GROUP = 128; //每一组有128块

public const uint SB\_DISK\_START = 0; //超级块区磁盘起始块号

public const uint USER\_DISK\_START = 3; //用户信息区磁盘起始块号

public const uint RECYCLEBINMAP\_DISK\_START = 10; //回收站Map表区磁盘起始块号

public const uint iNODE\_DISK\_START = 100; //i节点区磁盘起始块号

public const uint DATA\_DISK\_START = 4000; //数据区磁盘起始块号

public uint free\_block\_num = 1023 \* 1024 \* 4; //空闲块的数量,4GB-4MB

public uint free\_inode\_num = 1024 \* 50; //空闲i节点的数量

public uint last\_group\_block\_num = BLOCK\_IN\_GROUP;//最后一组的块的数量

public List<uint> last\_group\_addr; //最后一组的块的地址(保留区大小为4000块)

public bool change\_flag = true; //超级块修改标志

public uint max\_inode\_id = 100; //当前分配的最大i节点ID,因为有默认文件夹,所以第一次取个大一点的数值

public uint check\_byte = 707197; //校验位

}

i节点

public class DiskiNode

{

public uint id; //磁盘i节点ID

public string name; //文件(夹)名

public uint size; //文件(夹)大小

public Dictionary<uint, uint> uid; //用户ID，1~1000用户组1，2~2000用户组2...(uid,priority)

public List<uint> next\_addr = new List<uint>();//文件的磁盘块地址或者文件夹下的文件(夹)的i节点ID

public uint fore\_addr; //上层目录的i的ID

public DateTime t\_create; //文件(夹)创建时间

public DateTime t\_revise; //文件(夹)修改时间

public ItemType type; //类型：文件/文件夹

}

用户：

public class MemoryUser

{

//建议：用户数小于10，每个用户打开文件数小于40

public uint uid; //用户uid

public uint current\_folder; //当前所在文件夹的i节点ID

public List<uint> open\_file = new List<uint>();//用户打开文件表

public string newpassword; //用户新密码

}

组长块：

public class BlockLeader

{

public uint next\_blocks\_num; //下一组的磁盘块数量

public List<uint> block\_addr = new List<uint>();//下一组的每一块的地址

}

多次间址：

//虚拟间址寻址结构

public abstract class Data

{

public abstract int getItem(uint i = 0, uint j = 0);

}

//一次间址寻址结构实现

public class DataInt : Data

{

int adata;

public DataInt(int a)

{

adata = a;

}

public override int getItem(uint i = 0, uint j = 0)

{

return adata;

}

}

//二次间址寻址结构实现

public class DataIntArray : Data

{

public int[] data = new int[24];

public override int getItem(uint i = 0, uint j = 0)

{

return data[i];

}

public DataIntArray(int key, int value)

{

data[key] = value;

}

}

//三次间址寻址结构实现

public class DataIntArrayArray : Data

{

public int[,] data = new int[24, 24];

public override int getItem(uint i = 0, uint j = 0)

{

return data[i, j];

}

public DataIntArrayArray(int key1, int key2, int value)

{

data[key1, key2] = value;

}

}