数据库技术

张海宁

贵州大学

hnzhang1@gzu.edu.cn

April 5, 2018

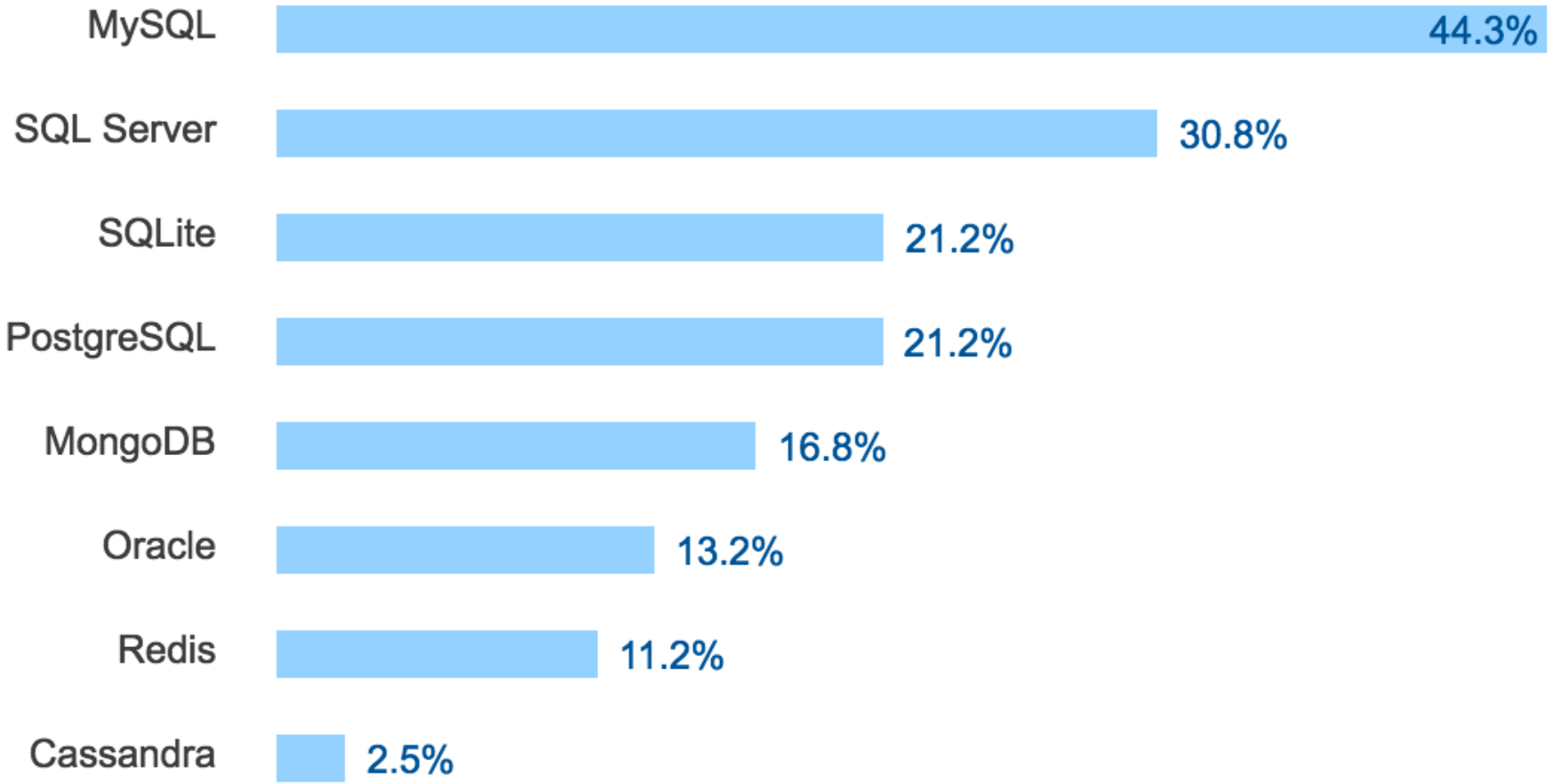
Overview

MySQL

Many of the world's largest and fastest-growing organizations including Facebook, Google, Adobe, Alcatel Lucent and Zappos rely on MySQL to save time and money powering their high-volume Web sites, business-critical systems and packaged software.

database popularity

Most popular databases in 2017 according to StackOverflow survey



https://www.eversql.com/
most-popular-databases-in-2017-according-to-stackoverflow-sur

分析

1. The most popular database is MySQL, and not by far comes SQL Server. Almost half of the developers who answered the survey (44.3% out of 36,935 responders) are using MySQL. It seems RDBMS databases and specifically MySQL are not going anywhere anytime soon.
2. RDBMS databases are still significantly more common than NoSQL databases such as MongoDB.
3. Relatively new technologies are starting to gain market share in the databases world –Redis (first release at 2009) and Cassandra (first release at 2008).
4. Almost 1/4 of all programmers (23.3%) are using SQLite, which is a lite SQL database which is based on a single file. This small database software is gaining popularity among developers, probably mostly for simple and standalone applications.

download and install MySQL

Download MySQL and MySQL connectors

`https://dev.mysql.com/downloads/workbench/`(GUI 工具)
`https://dev.mysql.com/downloads/mysql/`
`https://dev.mysql.com/downloads/connector/j/`

Install

一般地，对于 MySQL 和 MySQL workbench 默认下一步安装即可。MySQL Connector/J 解压，将 mysql-connector-java-xxx-bin.jar 文件加入到 eclipse 中项目的库文件路径中。(在 eclipse 中，右键项目-> 属性-> 库-> 添加外部 jar 文件)

JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.
JDBC 是 oracle 公司提出的一个标准，其具体实现由各家数据库公司自己进行。

JDBC 的功能

1. 同数据库建立连接
2. 向数据库发送 sql 语句
3. 处理从数据库返回的结果

JDBC 中常用的接口

接口	作用
Driver	负责加载驱动
DriverManager	负责管理驱动和连接数据库
Connection	管理某个数据库连接
Statement	执行 sql 语句，并返回结果
ResultSet	获得检索结果

Table: JDBC 的常用接口

连接数据库的一般流程

jsp 连接数据库的一般流程如下：

1. 加载 jdbc 驱动程序
2. 创建数据库连接
3. 执行 sql 语句
4. 获取查询结果
5. 关闭连接

加载 jdbc 驱动

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<%
    try{

        Class.forName("com.mysql.jdbc.Driver");

    }catch(ClassNotFoundException e){
        System.out.println("驱动加载失败！");
        e.printStackTrace();
    }
%>
```

建立数据库连接

使用DriverManager类的 getConnection() 方法建立 sql 连接。

```
<%
    try{
        Class.forName("com.mysql.jdbc.Driver");

        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost","root","password");

    }catch(Exception e){
        System.out.println("出现错误，具体内容如下：");
        e.printStackTrace();
    }
%>
```

执行 sql 语句

使用 Connection 类的对象conn 的 createStatement() 方法创建 Statement 类的对象st，再使用 st 对象来执行 sql 语句 (executeQuery() 或 executeUpdate())。

```
<%
    try{
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost/","root","password");
        Statement st = conn.createStatement();
        st.executeQuery("show databases;");
}catch(Exception e){
        System.out.println("出现错误，具体内容如下：");
        e.printStackTrace();
}
%>
```

获取查询结果并关闭数据库连接

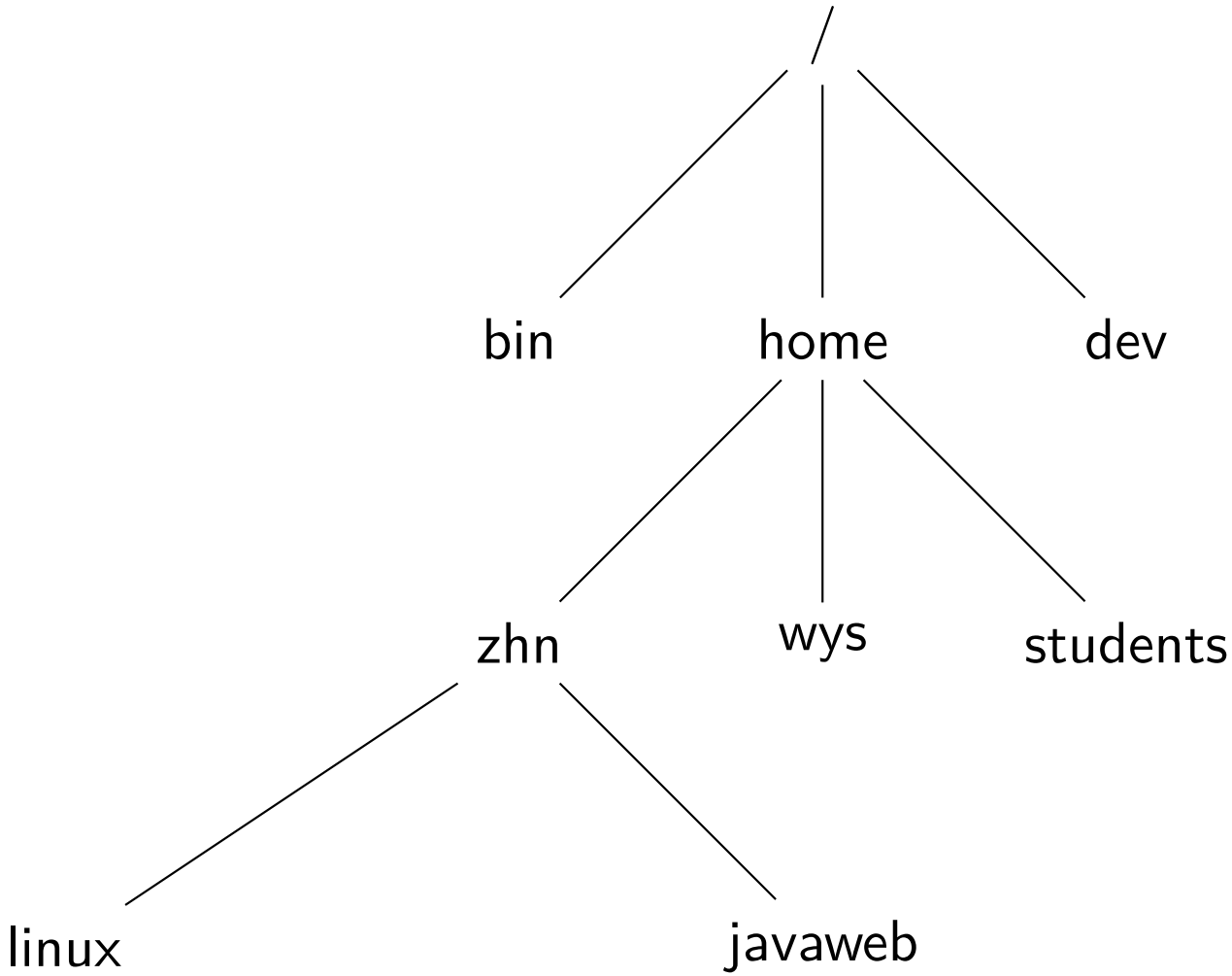
通过 Statement 类的对象st来执行 sql 语句 (executeQuery() 或 executeUpdate())，会返回一个 ResultSet 类的对象rs，rs 对象拥有一些方法可以获取查询到的值。

```
<%
try{
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn = DriverManager.getConnection(
     "jdbc:mysql://localhost/","root","password");
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("show databases;");
    while(rs.next()){
    String str = rs.getString(1);
    System.out.println(str);
}
     conn.close();
}catch(Exception e){ e.printStackTrace(); }
%>
```

完整 java 代码

```
<%
 try{
     Class.forName("com.mysql.jdbc.Driver");
     Connection conn = DriverManager.getConnection(
      "jdbc:mysql://localhost/","root","password");
     Statement st = conn.createStatement();
     ResultSet rs = st.executeQuery("show databases;");
     int column = rs.getMetaData().getColumnCount();
     System.out.println(column);
     while(rs.next()){
       System.out.println(rs.getString(1));
     }
      conn.close();
 }catch(Exception e){ e.printStackTrace(); }
%>
```

目录



Invoke system calls

It is not possible to directly link user-space applications with kernel space. For reasons of security and reliability, user-space applications must not be allowed to directly execute kernel code or manipulate kernel data. Instead, the kernel must provide a mechanism by which a user-space application can "signal" the kernel that it wishes to invoke a system call. The application can then trap into the kernel through this well-defined mechanism, and execute only code that the kernel allows it to execute. The exact mechanism varies from architecture to architecture.

The C Library

The C library (libc) is at the heart of Unix applications. Even when you're programming in another language, the C library is most likely in play, wrapped by the higher-level libraries, providing core services, and facilitating system call invocation. On modern Linux systems, the C library is provided by GNU libc, abbreviated glibc, and pronounced gee-lib-see or, less commonly, glib-see.

The GNU C library provides more than its name suggests. In addition to implementing the standard C library, glibc provides wrappers for system calls, threading support, and basic application facilities.

The C Compiler

In Linux, the standard C compiler is provided by the GNU Compiler Collection (gcc). Originally, gcc was GNU's version of cc, the C Compiler. Thus, gcc stood for GNU C Compiler. Over time, support was added for more and more languages. Consequently, nowadays gcc is used as the generic name for the family of GNU compilers. However, gcc is also the binary used to invoke the C compiler. In this course, when we talk of gcc, we typically mean the program gcc, unless context suggests otherwise.

用户程序 vs 库函数 vs 系统调用

Versions and Help

We can display the version of GCC with –version/-v option, and get help with –help option:

Example (show version and help of GCC)

```
$ gcc -v
$ gcc --version
$ gcc --help
```

编写 c 程序的流程

编写 c 程序的流程

hello.c

Example

```
// hello.c
#include <stdio.h>

int main(){
        printf("Hello, C.\n");
        return 0;
}
```

从.c 文件到可执行文件

预处理

gcc -E hello.c -o hello.i

编译

gcc -S hello.i

汇编

gcc -c hello.s -o hello.o

链接

gcc hello.o -o hello.out

all in one

gcc hello.c -o hello.out

静态库和动态库

库文件是一组被预先编译的目标文件的集合，可以被链接到用户所编写的程序中。

- 静态库
  通常以“.a”结尾 (archive file)，静态库的代码在编译时就会连接到用户的程序中
- 动态库
  通常以“.so”结尾 (shared objects)，用户的程序在运行时，按需要载入

几个有用的工具

1. file
   判断文件的类型
2. nm
   显示目标文件的符号表，通常用于查看目标文件中是否定义了某个特定的函数
3. ldd
   检查一个可执行文件并显示出其需要的动态库文件
4. mtrace
   内存溢出检测

make

依照 makefile 文件，make 程序可以自动确定一个软件包的哪些部分需要重新编译。
makefile 文件描述了建立可执行程序的一些规则。

makefile

hello.c

```c
// hello.c
#include <stdio.h>

int main(){
    printf("Hello,C.\n");
    return 0;
}
```

makefile

```
all:hello

hello:hello.o
	gcc -o hello hello.o

hello.o:hello.c
	gcc -c hello.c
clean:
	rm hello.o
```

makefile 的规则

```
target: pre-req-1 pre-req-2 ...
^^Icommand
```

The target and pre-requisites are separated by a colon (:). The command must be preceded by a tab (NOT spaces).

make 和 makefile

```
$ make
gcc -c hello.c
gcc -o hello hello.o
$ make clean
rm hello.o
$ ls
hello   hello.c   makefile
```

修改源文件与 make 的关系

```
$ make
gcc -c hello.c
gcc -o hello hello.o
$ make
make: Nothing to be done for 'all'.
$ vim hello.c
$ make
gcc -c hello.c
gcc -o hello hello.o
$ ./hello
Hello, C.
test make.
```

GDB

GDB(GNU Source-Level Debugger)，是 GNU 的一个命令行调试工具。
它主要可以完成以下功能：

- 设置运行环境和参数运行指定程序
- 让程序在指定的条件下停止
- 当程序停止时，检查发生了什么
- 改变正在调试的程序，以修正某个 bug，然后继续调试

gdb 部分命令

命令	作用	命令	作用
file	载入可执行文件	list	显示源代码
run	执行	info local	显示当前函数中变量值
kill	停止执行	info break	显示断点列表
break	设置断点	info func	显示所有函数名
delete	删除断点	watch	监视表达式的变化
print	显示表达式的值	next	执行下一条代码，但不进入
shell 命令	执行 shell 命令	step	执行下一条代码，进入函数

Table: GDB 部分命令

gdb 示例

```
$ vim gdb.c
$ cat gdb.c
#include <stdio.h>
int main(){
        int i=0;
        for(i=0;i<=15;i++){
                printf("the number is %d",i);
        }
}
$ gcc -ggdb -o gdbTest gdb.c
```

gdb 示例

总结

1. System Call,C Compiler,C Library
2. GCC
3. make,makefile
4. GDB

作业

按以下要求进行编程练习：

1. 2 个 cpp 文件：prog.cpp，aux.cpp
2. 1 个 h 文件：aux.h
3. aux.h 头文件定义函数 Max 和 Min，它们分别计算四个数（参数）的最大值和最小值；aux.cpp 实现这两个函数
4. prog.c 中定义主函数，循环输入 4 个随机数，输出他们的最大和最小值
5. 编译该项目，调试、跟踪程序执行过程；并在控制台界面运行编译的可执行文件。
6. 编写类似功能的 c 语言程序也可。

The End