

1. 监听EasyMall网站的日志信息

在EasyMall网站中有很多用户的操作过程，可以将这些用户的操作过程进行记录，记录后的数据可以为后期的数据处理分析提供使用。

2. 收集网站中的日志信息

监听EasyMall这个web应用的创建 --- ServletContextListener

a. 代码实现:

```
package com.easymall.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
//监听EasyMall应用启动和销毁的过程
public class MyServletContextListener implements ServletContextListener {

    //ServletContext对象产生
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("EasyMall应用启动.");
    }

    //ServletContext对象销毁
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("EasyMall应用销毁.");
    }

}
```

i. 配置信息:

```
<!-- 网站的监听器 -->
<!-- EasyMall应用监听 -->
<listener>
    <listener-class>com.easymall.listener.MyServletContextListener</listener-class>
</listener>
```

监听用户登录和注销的过程 --- HttpSessionBindingListener

```
package com.easymall.domain;

import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class User implements HttpSessionBindingListener{
    private int id;
    private String username;
    private String password;
    private String nickname;
    private String email;

    public User() {
        super();
    }
}
```

```

public User(int id, String username, String password, String nickname,
String email) {
    super();
    this.id = id;
    this.username = username;
    this.password = password;
    this.nickname = nickname;
    this.email = email;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getUsername() {
    return username;
}
public void setUsername(String username) {
    this.username = username;
}
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
}
public String getNickname() {
    return nickname;
}
public void setNickname(String nickname) {
    this.nickname = nickname;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
//监听javabean在session域中的添加过程
//登录EasyMall
public void valueBound(HttpSessionBindingEvent event) {
    System.out.println("用户["+username+"]已经登录EasyMall.");
}
//监听javabean在session域中的移除过程
//注销EasyMall
public void valueUnbound(HttpSessionBindingEvent event) {
    System.out.println("用户["+username+"]已经注销EasyMall.");
}
}
}

```

监听用户发送的请求 --- ServletRequestListener

i. 代码实现:

```
package com.easymall.listener;
```

```

import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;

import com.easymall.domain.User;

public class MyServletRequestListener implements ServletRequestListener {

    //用户请求结束
    public void requestDestroyed(ServletRequestEvent sre) {
        HttpServletRequest request = (HttpServletRequest) sre.getServletRequest();
        //用户未登录状态的username值
        String username = "游客";
        //用户名
        //判断当前请求中是否包含用户的登录信息
        if(request.getSession(false) != null
            && request.getSession().getAttribute("user") != null){
            User user = (User) request.getSession().getAttribute("user");
            username = user.getUsername();
        }

        //用户ip
        String ip = request.getRemoteAddr();
        //用户请求的url地址
        String url = request.getRequestURL().toString();
        System.out.println("用户["+username+"] | ip:"+ip+" | url:"+url+"访问结束");

    }

    //用户请求开始
    public void requestInitialized(ServletRequestEvent sre) {
        HttpServletRequest request = (HttpServletRequest) sre.getServletRequest();
        //用户未登录状态的username值
        String username = "游客";
        //用户名
        //判断当前请求中是否包含用户的登录信息
        if(request.getSession(false) != null
            && request.getSession().getAttribute("user") != null){
            User user = (User) request.getSession().getAttribute("user");
            username = user.getUsername();
        }

        //用户ip
        String ip = request.getRemoteAddr();
        //用户请求的url地址
        String url = request.getRequestURL().toString();
        System.out.println("用户["+username+"] | ip:"+ip+" | url:"+url+"访问开始");

    }

}

```

ii. 配置信息:

```

<!-- 网站请求监听请求 -->
<listener>
    <listener-class>com.easymall.listener.MyServletRequestListener</listener-class>
</listener>

```


1. Log4j是什么

log for java 在大部分的java程序都现在都已经使用log4j日志框架来收集日志。原因是log4j使用简单，操作便捷，非常善于日志的收集。

2. Log4j的使用方式：

- 编写一个log4j的配置文件。放入src目录。
- 在类中添加一个Logger类型的对象，通过这个对象记录日志。
- 导入log4j的jar包。

3. 一个日志文件组成

日志输出级别。

设置日志的输出级别，用来区分当前日志信息的重要程度，日志对程序影响的越严重日志级别越高。如果用户指定日志输出级别，则在日志中输出当前指定级别及其以上级别的日志信息。(注意：根中可以对日志级别做出全局的规定。)

off 最高等级，用于关闭所有日志记录。
fatal 指出每个严重的错误事件将会导致应用程序的退出。
error 指出虽然发生错误事件，但仍然不影响系统的继续运行。
warn 表明会出现潜在的错误情形。
info 一般和在粗粒度级别上，强调应用程序的运行全程。
debug 一般用于细粒度级别上，对调试应用程序非常有帮助。
all 最低等级，用于打开所有日志记录

日志输出的位置。

org.apache.log4j.ConsoleAppender (控制台) ,
org.apache.log4j.FileAppender (文件) ,
org.apache.log4j.DailyRollingFileAppender (每天产生一个日志文件) ,
org.apache.log4j.RollingFileAppender (文件大小到达指定尺寸的时候产生一个新的文件) ,
org.apache.log4j.WriterAppender (将日志信息以流格式发送到任意指定的地方)

日志输出的格式。

设置日志的格式是用来指定数据以何种形式输出到控制台或文件中。

%p 输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL
%r 输出自应用启动到输出该log信息耗费的毫秒数
%c 输出所属的类目，通常就是所在类的全名
%t 输出产生该日志事件的线程名
%n 输出一个回车换行符，Windows平台为“rn”，Unix平台为“n”
%d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyy MMM dd HH: mm: ss, SSS}，输出类似
：2002年10月 18日 22：10：28，921
%l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.
main(TestLog4.java: 10)
%m所需输出的日志信息

布局格式：

```
org.apache.log4j.HTMLLayout (以HTML表格形式布局) ,
org.apache.log4j.PatternLayout (可以灵活地指定布局模式) ,
org.apache.log4j.SimpleLayout (包含日志信息的级别和信息字符串) ,
org.apache.log4j.TTCCLayout (包含日志产生的时间、线程、类别等等信息)
```

4. 使用Log4j

- a. 创建一个Logger对象，通过Logger对象引出对应级别的日志方法。

```
#log4j的根，          日志输出级别，各个管道的名称，如果管道名称存在，则必须制定管道对应的配置内容
log4j.rootLogger = debug,stdout,D,E

log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target = System.out
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%m%n

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender
log4j.appender.D.File = D://logs/log.log
log4j.appender.D.Append = true
log4j.appender.D.Threshold = DEBUG
log4j.appender.D.layout = org.apache.log4j.PatternLayout
log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

log4j.appender.E = org.apache.log4j.DailyRollingFileAppender
log4j.appender.E.File = D://logs/error.log
log4j.appender.E.Append = true
log4j.appender.E.Threshold = ERROR
log4j.appender.E.layout = org.apache.log4j.PatternLayout
log4j.appender.E.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n
```

获取日志信息：代码实现

```
package cn.tedu.log4j;

import org.apache.log4j.Logger;

public class Demo1 {

    public static Logger logger = Logger.getLogger(Demo1.class);
    public static void main(String[] args) {
        logger.debug("调试级别");
        logger.info("普通日志信息");
        logger.warn("警告日志信息");
        logger.error("错误日志信息");
        logger.fatal("致命的日志信息");
    }

}
```

5. 在EasyMall中引入log4j

代码实现：各个类中分别创建logger对象，调用对象身上的日志级别方法

```
package com.easymall.domain;

import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;
```

```

import org.apache.log4j.Logger;

public class User implements HttpSessionBindingListener{
    private int id;
    private String username;
    private String password;
    private String nickname;
    private String email;

    public static Logger logger = Logger.getLogger(User.class);
    public User() {
        super();
    }
    public User(int id, String username, String password, String nickname,
        String email) {
        super();
        this.id = id;
        this.username = username;
        this.password = password;
        this.nickname = nickname;
        this.email = email;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getNickname() {
        return nickname;
    }
    public void setNickname(String nickname) {
        this.nickname = nickname;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    //监听javabean在session域中的添加过程
    //登录EasyMall
    public void valueBound(HttpSessionBindingEvent event) {
//        System.out.println("用户["+username+"]已经登录EasyMall.");
        logger.error("用户["+username+"]已经登录EasyMall.");
    }
}

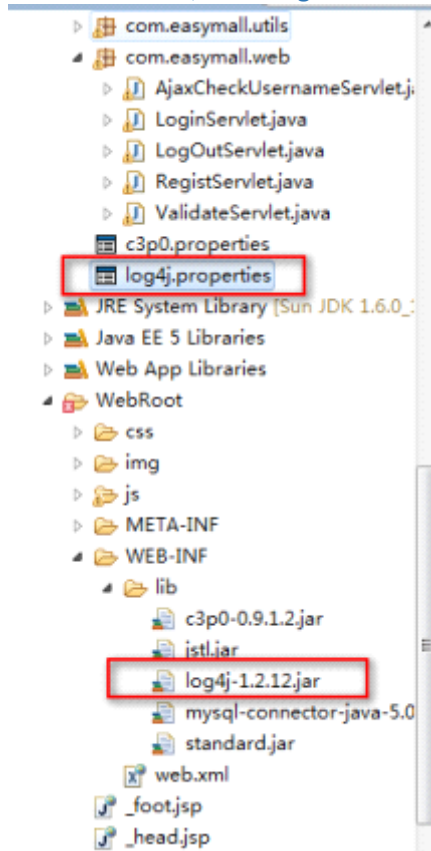
```

```

    }
    //监听javabean在session域中的移除过程
    //注销EasyMall
    public void valueUnbound(HttpSessionBindingEvent event) {
//        System.out.println("用户["+username+"]已经注销EasyMall.");
        logger.fatal("用户["+username+"]已经注销EasyMall.");
    }
}
}

```

添加配置文件，添加log4j. jar包



1. 事务的概念

事务是一个不可分割的单位，包含一组操作，这组操作要么全部成功，要么全部失败。

案例:转账

```
update user set money = money -100 where name='a';
```

```
update user set money = money +100 where name = 'b';
```

2. 数据库中实现事务

以前执行sql语句时，并为开启事务，是因为在数据库中默认情况下一个sql语句就具有一个事务，sql语句执行会对数据库中的数据做出真实修改，即使尝试撤销操作也无法恢复已经修改过的数据。

start transaction;	开启事务。如果事务开启，则在rollback或commit之前的全部sql语句都是事务的一部分。这些内容要么全成功，要么全失败。
rollback;	回滚事务。将已经修改但是并未提交的数据恢复到之前的内容，这个过程称之为回滚。
commit;	提交事务。如果事务被提交，则数据库中的数据就真实的发生的改变无法撤销这个过程。

3. 在JDBC中实现事务的控制。

conn.setAutoCommit(boolean)	设置自动提交.默认状态下参数值为true，代表开启自动提交，本质是表示一个sql语句就是一个事务。如果设置为false则表示关闭自动提交，开启事务。
conn.commit();	提交事务。
conn.rollback();	回滚事务。
conn.rollbackc(SavePoint)	设置一个保存点。可以经事务回滚到保存点的位置。回滚到保存点后再次对保存点之前的内容选择提交或继续回滚的操作。

代码实现demo1:

```
package cn.tedu.trans;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

//JDBC实现事务的使用

```

public class TransDemo1 {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
            //1.注册数据库驱动
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建连接
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/day17", "root", "root");
            //关闭自动提交，开始事务
            conn.setAutoCommit(false);

            ps = conn.prepareStatement("update user set money =money -100 where name = ?");
            ps.setString(1, "a");
            ps.executeUpdate();

            int i=1/0;

            ps = conn.prepareStatement("update user set money =money +100 where name = ?");
            ps.setString(1, "b");
            ps.executeUpdate();
            conn.commit();

        } catch (Exception e) {
            e.printStackTrace();
            if(conn != null){
                try {
                    conn.rollback();
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }
        }

        finally{
            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}

```

代码实现Demo2：（保存点）

```

package cn.tedu.trans;

```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Savepoint;

//JDBC实现事务的使用
public class TransDemo2 {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        Savepoint sp = null;
        try {
            //1.注册数据库驱动
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建连接
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/day17", "root", "root");
            //关闭自动提交，开始事务
            conn.setAutoCommit(false);

            ps = conn.prepareStatement("update user set money =money -100 where name = ?");
            ps.setString(1, "a");
            ps.executeUpdate();

            ps = conn.prepareStatement("update user set money =money +100 where name = ?");
            ps.setString(1, "b");
            ps.executeUpdate();
            //保存点：保存点之前的操作全部成功，保存点之后的数据全部失败。
            sp = conn.setSavepoint();

            int i=1/0;

            ps = conn.prepareStatement("update user set money =money -100 where name = ?");
            ps.setString(1, "a");
            ps.executeUpdate();

            ps = conn.prepareStatement("update user set money =money +100 where name = ?");
            ps.setString(1, "b");
            ps.executeUpdate();

            conn.commit();

        } catch (Exception e) {
            e.printStackTrace();
            if(conn != null){
                if(sp != null){
                    try {
                        //如果sq不为null，则回滚到保存点
                        conn.rollback(sp);
                        //保存点之前的SQL需要被提交
                        conn.commit();
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }
                }
            }
        }
    }
}

```

```
        }  
    }  
}  
  
}finally{  
    if(conn != null){  
        try {  
            conn.close();  
        } catch (SQLException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
}  
}  
}
```

1. 事务四大特性 (ACID)

原子性(Atomicity)

原子性表示事务是一个不可分割的单位，这个单位内的所有操作，要么全部成功，要么全部失败。

一致性(Consistency)

一致性表示事务修改前后的数据在完整性上是一致的。

隔离性(Isolation)

隔离性是为数据库线程安全问题提出的，在一个事务执行的过程中，可能被另一个事务所影响，应该控制两个事务之间是相互独立的。

持久性(Durability)

持久性表示数据库中的事务一旦修改了数据，这个修改的过程就是真实的发生，数据库中的数据也真实的发生的改变。任何人都无法撤销，改变后的数据。

2. 隔离性

在数据库的事务中，数据库厂商已经对事务的原子性、一致性和持久性进行的实现，只保留隔离性没有做出明确的实现，是因为用户在面对线程安全问题的时候，可能会有不同的需求。

情景分析：

- i. 两个线程都是读操作，是否会发生线程安全问题？
不会。因为两个读操作不会读数据进行修改，无论如何读取数据，读取的结果都不会发生改变。所以不会出现线程安全问题。
- ii. 两个线程都是写操作，是否会发生线程安全问题？
会。合理的添加数据库中的锁，可以解决线程安全问题。
- iii. 一个读线程，一个写线程，是否会发生线程安全问题？
在面对一个读线程一个，写线程的时候，可能会发生线程安全问题，如果这个问题出现，是否解决这个问题需要再次讨论。

分析结果：

如果两个线程都为读则不会发生线程安全问题，无需修改任何数据库的隔离级别问题。

如果两个线程都为写则一定会发生线程安全问题，这时必须要修改数据库的隔离级别为serializable，才能避免线程安全问题。

如果一个读，一个写线程，则可能会发生线程安全问题，所发生的线程安全问题可以选择解决，也可以不解决，取决于具体的开发情景。如果需要解决数据库也提供了四种数据库的隔离级别，可以手动设置这些隔离级别。

在面对一个读，一个写线程时，可能出现的问题有：

脏读、不可重复读、虚读/幻读

3. 情景一：

脏读：一个事务读取到另一个未提交事务的数据，导致事务提交前后的查询结果不一致。

a为买家 b为店主, a从b店中买鞋子。

a 1000

b 1000

```
a:  start transaction;
    update user set money =money -100  where name='a';
    update user set money = money +100  where name='b';
```

b: start transaction;
 select * from user where name = 'b';
 b ---1100

a: rollback;

b:
select * from user where name = 'b';
b-- 1000

4. 情景二

不可重复读：一个事务读取到了另外一个已经提交的事务，对表中的数据产生了影响，在事务执行前后查询的数据结果不一致。

a为银行客户, b为银行工作人员, b要查询a的账户总额。

a 1000 1000 1000

b: start transaction;
 select 活期 from account where name = 'a'; --- 1000
 select 定期 from account where name = 'a' --- 1000
 select 股票 from account where name = 'a' --- 1000

a:
start transaction;
update account set 活期 = 活期- 100 where name = 'a' ; ----- 900
commit;

b:
select 活期+定期+股票 from account where name = 'a' ; ----- 2900
commit;

5. 情景三

虚读/幻读：一个事务读取到另一个事务已经被提交的数据，被提交的数据对整张表的结果产生的了影响。

a 1000

```

b      2000
-----

c:
  start transaction;
  select sum(money) from user; --- 总额 3000
  select count(name) from user; --- 总人数 2
-----

d:
  start transaction;
  insert into user values (3, 'd', 3000); --- d存入3000
  commit;
-----

c:
  select avg(money) from user; --- 2000;
  commit;
-----

```

6. 数据库的隔离级别

read uncommitted	不管理用户任何的操作，用户任意的读和写，如果出现线程安全问题，也不会做任何处理。在此隔离级别下，脏读、不可重复读、虚读都可能会出现。
read committed	会对用户的读、写线程进行管理，但是只能避免脏读，不能避免不可重复读和虚读。
repeatable read	会对用户的读、写线程进行较强的管理，可以避免脏读和不可重复读，但是不能避免虚读。 数据库的默认隔离级别
serializable	将数据库修改为一个 串行化的工作方式 ，这种方式下只能允许一个线程操作数据库中的某一张表，由于将多线程强制修改为单线程操作，所以一定不会出现多线程并发安全问题。这种隔离级别可以避免脏读，不可重复读，虚读/幻读。

a. 从性能角度排序：

read uncommitted > read committed > repeatable read > serializable

b. 从安全角度排序：

serializable > repeatable read > read committed > read uncommitted

- c. 在数据库中的隔离性，一般不会采用read uncommitted, 因为安全性太差，也不会采用serializable, 因为效率太低。普遍会使用repeatable read 或者read committed两者之一。这两者读写效率适中，安全性适中，比较适合大部分的开发情景。

d. 查看数据库的隔离级别：

```
select @@tx_isolation;
```

e. 设置数据库的隔离级别

```
set global/session transaction isolation level read uncommitted;
```

设置数据库的隔离级别，当前sql语句中可以使用global来定义全局的隔离级别，由于定义之后是对服务器的配置进行修改，所以需要重启mysql客户端。如果使用session则定义当前会话范围内的隔离级别，只在当前窗口。如果不写global/session，则表示在下一次的事务当中定义隔离级别。

7. 数据库中的锁

数据库中有很多锁：

包括共享锁和排他锁。

在非serializable级别下，查询不添加任何锁。

在serializable级别下，查询会添加共享锁。写操作会添加排他锁。

a. 特性：

共享锁和共享锁可以共存。

共享锁和排他锁不能共存。

排他锁和任何锁都不能共存。

- b. 如果有共享锁和排他锁同时出现，则后出现的锁需要等待前一个事务的锁释放，才可以继续执行当前事务。

c. 死锁

i. 死锁出现的情况：

两个或多个线程相互等待对象释放资源，造成死锁的情况。

ii. 解决死锁：

1) 任意杀死一个线程，另外一个线程就可以正常执行。

2) 优化锁方案。将锁住部分的代码进行修改，避免死锁的出现。

8. 行级锁和表级锁

行级锁：表示在数据库表当中为某一行添加锁。当前被锁住的行只允许一个线程操作。

表级锁：表示在数据库中，为选中的表添加一个表级锁，在一个线程操作当前表的状态下，不允许另外一个线程操作选中的表。