[Eclipse中集成Gitee插件 2](#_Toc529366636)

[版本控制系统介绍 2](#_Toc529366637)

[Git 2](#_Toc529366638)

[Lock 2](#_Toc529366639)

[Condition 3](#_Toc529366640)

[阻塞队列 4](#_Toc529366641)

[BlockQueue 4](#_Toc529366642)

[装饰模式 5](#_Toc529366643)

[JSON解析 7](#_Toc529366644)

[原生Json解析 7](#_Toc529366645)

[使用FastJson进行Json解析 7](#_Toc529366646)

[XML解析 7](#_Toc529366647)

[DOM4J的使用 7](#_Toc529366648)

[多线程断点下载实现 7](#_Toc529366649)

[课堂练习 7](#_Toc529366650)

[家庭作业 9](#_Toc529366651)

[参考 10](#_Toc529366652)

# Eclipse中集成Gitee插件

## 版本控制系统介绍

开发必备的一项工具.就像时光穿梭机一样可以带你回到从前.(回到之前的任意一次提交)

## Git

常见的版本控制有:SVN.Git.Git的使用率很高.

# Lock

1. /\*\*
2. \*
3. \*/
4. **package** com.hwua.dmeo01;
6. **import** java.util.concurrent.locks.Lock;
7. **import** java.util.concurrent.locks.ReentrantLock;
9. /\*\*
10. \* @author Administrator
11. \*
12. \*/
13. **public** **class** Demo02 {
15. **private** Lock lock = **new** ReentrantLock();
17. **public** **static** **void** main(String[] args) {
18. **final** Demo02 d = **new** Demo02();
20. **new** Thread() {
21. @Override
22. **public** **void** run() {
23. d.test(**this**);
24. }
25. }.start();
27. **new** Thread() {
28. @Override
29. **public** **void** run() {
30. d.test(**this**);
31. }
32. }.start();
33. }
35. **public** **void** test(Thread t) {
36. // -- 当返回值为true的时候就代表已经获取到锁了
37. **if** (lock.tryLock()) {
38. System.out.println(t.getName() + "获取锁!");
39. **try** {
40. Thread.sleep(3000);
41. } **catch** (InterruptedException e) {
42. // TODO Auto-generated catch block
43. e.printStackTrace();
44. } **finally** {
45. System.out.println(t.getName() + "释放锁!");
46. lock.unlock();
47. }
48. } **else** {
49. System.out.println(t.getName() + "获取锁失败!");
50. }
51. }
53. }

## Condition

Condition是一个信号.用于表示对共有资源的占有和释放.通过Lock对象获取.

# 阻塞队列

最近流行一些消息中间件比如:MQ它们的核心都是阻塞队列.

简单说是把等待放到放到一个队列中.而队列具备先进先出的特点.除此以外队列还有特点.空了没得出,满了没得进.

## BlockQueue

1. /\*\*
2. \*
3. \*/
4. **package** com.hwua.dmeo01;
6. **import** java.util.concurrent.ArrayBlockingQueue;
7. **import** java.util.concurrent.BlockingQueue;
9. /\*\*
10. \* @author Administrator
11. \*
12. \*/
13. **public** **class** Demo04 {
14. **public** **static** **void** main(String[] args) {
15. **final** BlockingQueue<Character> dq = **new** ArrayBlockingQueue<>(10);
17. // -- 假设它是生成者
18. **new** Thread() {
19. **public** **void** run() {
20. **for** (**char** i = 'A'; i <= 'Z'; i++) {
21. **try** {
22. dq.put(i);
23. System.out.println(getName() + "已经生产一个商品+" + i + ",库存数量:" + dq.size());
24. Thread.sleep(500);
25. } **catch** (InterruptedException e) {
26. // TODO Auto-generated catch block
27. e.printStackTrace();
28. }
29. }
30. };
31. }.start();
33. // -- 假设它是消费者
34. **new** Thread() {
35. **public** **void** run() {
36. **char** ele = 0;
37. **while** (**true**) {
38. **try** {
39. ele = dq.take();
40. Thread.sleep(1000);
41. } **catch** (InterruptedException e) {
42. // TODO Auto-generated catch block
43. e.printStackTrace();
44. }
45. System.out.println(getName() + "消费一个商品" + ele + ",库存数量:" + dq.size());
46. }
47. };
48. }.start();
49. }
50. }

# 装饰模式

在动态代理那里一定要学会装饰.还要做到对比装饰和动态代理的区别.(因为它们的目标是一样的.)

引用Head Fist Design中关于装饰模式的案例:

在装饰模式中有四个角色：

1. 抽象构件（Component）角色：给出一个抽象接口，以规范准备接收附加责任的对象。
2. 具体构件（Concrete Component）角色：定义一个将要接收附加责任的类。
3. 抽象装饰（Decorator）角色：持有一个构件（Component）对象的实例，并实现一个与抽象构件接口一致的接口。
4. 具体装饰（Concrete Decorator）角色：负责给构件对象添加上附加的责任。

# JSON解析

Json是目前比较常见的用于多端通信使用的文本传输格式.

JSON:JavaScript Object Notation.是一种轻量级的数据交互格式.

有升级版(yaml.在Spring Boot中主流使用.)JSONP

JSON有两种格式:

1. 键值对
2. 值的有序列表(简单说就是数组)

## JSON的键

JSON的键只能是String类型

## JSON的值

JSON的值可以有很多类型

* 数值
* 字符串
* 对象(JSON对象.键值对)
* 数组(JSON数组.就是JSON的第二形态,值的有序列表)
* True
* False
* Null

## 原生Json解析

## 使用FastJson进行Json解析

使用FastJson解析天气预报

1. /\*\*
2. \*
3. \*/
4. **package** com.hwua.demo03;
6. **import** java.io.IOException;
7. **import** java.io.InputStream;
8. **import** java.net.MalformedURLException;
9. **import** java.net.URL;
10. **import** java.util.ArrayList;
11. **import** java.util.List;
13. **import** com.alibaba.fastjson.JSONArray;
14. **import** com.alibaba.fastjson.JSONObject;
16. /\*\*
17. \* @author Administrator
18. \*
19. \*/
20. **public** **class** Demo02 {
22. **public** **static** **void** main(String[] args) {
24. **final** String path = "http://api.k780.com/?app=weather.future&weaid=36&appkey=12897&sign=ad041abb874869a9764b5891234459b3&format=json";
25. List<Weather> weaList = parserWeather(getJsonText(path));
26. **for** (Weather weather : weaList) {
27. System.out.println(weather);
28. System.out.println("---------------------------");
29. }
31. }
33. **public** **static** List<Weather> parserWeather(String tet) {
34. // -- 解析JSON得到天气
35. ArrayList<Weather> list = **new** ArrayList<>();
36. // -- 1.有Text文本生成JSON对象
37. JSONObject obj = JSONObject.parseObject(tet);
38. // -- 2.读取Success键所代表的结果并判断是否为1.
39. **int** i = Integer.parseInt(obj.getString("success"));
40. **if** (i != 1) {
41. System.out.println("获取天气失败!" + obj.getString("msg"));
42. **return** **null**;
43. }
44. // -- 3.把含有详细天气的JSONArry获取出来
45. JSONArray array = obj.getJSONArray("result");
46. // -- 4.对数组进行遍历
47. **for** (**int** j = 0; j < array.size(); j++) {
48. // -- 获取数组中的每一个JSON对象
49. JSONObject weaObj = array.getJSONObject(j);
50. // -- Json对象转变成weather对象
51. Weather w = JSONObject.toJavaObject(weaObj, Weather.**class**);
52. list.add(w);
53. }
54. **return** list;
55. }
57. **private** **static** String getJsonText(String path) {
58. StringBuilder sb = **new** StringBuilder();
59. **try** {
60. URL url = **new** URL(path);
61. InputStream is = url.openStream();
62. **byte**[] buffer = **new** **byte**[1024];
63. **int** length = 0;
64. **while** (-1 != (length = is.read(buffer))) {
65. sb.append(**new** String(buffer, 0, length, "UTF-8"));
66. }
67. } **catch** (MalformedURLException e) {
68. e.printStackTrace();
69. } **catch** (IOException e) {
70. e.printStackTrace();
71. }
72. // -- System.out.println(sb.toString());
73. **return** sb.toString();
74. }
75. }

# XML解析

## DOM4J的使用

# 多线程断点下载实现

# 课堂练习

1. 使用Lock来打印ABABAB
2. /\*\*
3. \*
4. \*/
5. **package** com.hwua.dmeo01;
7. **import** java.util.concurrent.locks.Condition;
8. **import** java.util.concurrent.locks.Lock;
9. **import** java.util.concurrent.locks.ReentrantLock;
11. /\*\*
12. \* @author Administrator
13. \*
14. \*/
15. **public** **class** Demo03 {
16. **private** Lock lock = **new** ReentrantLock();
17. Condition conA = lock.newCondition();
18. Condition conB = lock.newCondition();
20. **public** **static** **void** main(String[] args) {
21. **final** Demo03 d = **new** Demo03();
22. **new** Thread() {
23. **public** **void** run() {
24. **while** (**true**) {
25. // -- 尝试获取锁
26. **if** (d.lock.tryLock()) {
27. **try** {
28. // -- 告诉B你可以开始准备了
29. d.conB.signal();
30. System.out.println("A");
31. **try** {
32. // -- 让A自己休眠.别写了 是await 如果是Object的则是wait和noftify
33. d.conA.await();
34. } **catch** (InterruptedException e) {
35. // TODO Auto-generated catch block
36. e.printStackTrace();
37. }
38. } **finally** {
39. // -- 一定要把解锁写在if中.只有 有锁才能解锁.
40. d.lock.unlock();
41. }
42. } **else** {
43. System.out.println(getName() + "没有获得锁!");
44. }
45. }
46. };
47. }.start();
49. **new** Thread() {
50. **public** **void** run() {
51. **while** (**true**) {
52. // -- 尝试获取锁
53. **if** (d.lock.tryLock()) {
54. **try** {
55. d.conA.signal();
56. System.out.println("B");
57. **try** {
58. d.conB.await();
59. } **catch** (InterruptedException e) {
60. // TODO Auto-generated catch block
61. e.printStackTrace();
62. }
63. } **finally** {
64. // -- 一定要把解锁写在if中.只有 有锁才能解锁.
65. d.lock.unlock();
66. }
67. } **else** {
68. System.out.println(getName() + "没有获得锁!");
69. }
70. }
72. };
73. }.start();
74. }
76. }
77. 有四个售票口卖100张票.使用阻塞队列来实现!

# 家庭作业

1. 麦当劳有三个服务台在进行点餐工作.后台有两名工作人员在制作汉堡.现在有ABCD四个客户来点餐分别点:海陆空三味大餐,无辣不欢组合,经典麦辣鸡腿堡套餐,不素之霸双牛堡套餐.使用线程池和阻塞队列来实现这一过程.
2. 孙悟空七十二变，用装饰，写出三变就可以

# 参考

1. [Lock](https://www.cnblogs.com/dolphin0520/p/3923167.html)
2. [Synchronized](https://www.cnblogs.com/dolphin0520/p/3923737.html)
3. [Volatile关键字](https://www.cnblogs.com/dolphin0520/p/3920373.html)
4. [阻塞队列](http://ifeve.com/java-blocking-queue/)
5. [装饰设计模式](https://blog.csdn.net/lissdy/article/details/7540090)