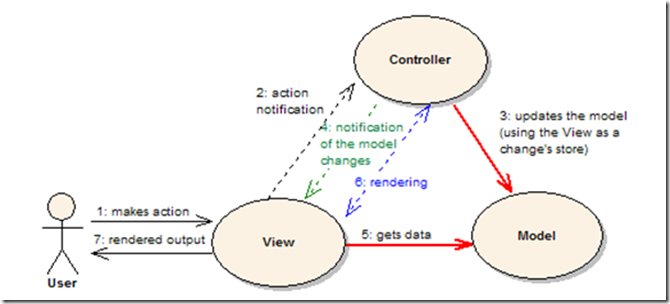
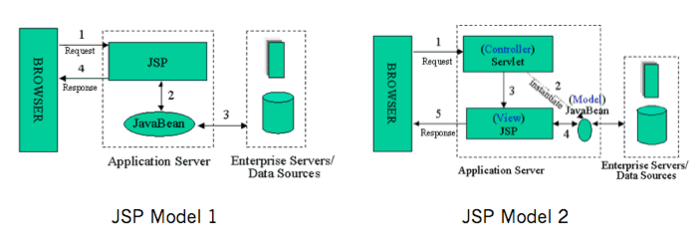
## MVC MVP MVVM

### MVC



1. 服务器通过路由规则把请求交给特定的Controller处理（也就是说可以有多个Controller），每一个View（JSP）向Controller注册成为一个Observer。
2. Controller（Servlet）接收传来的请求，负责转发到相应的JavaBean(Model)
3. JavaBean(Model)处理业务，更新数据。
4. Controller根据Model返回的结果跳转到相应的JSP
5. JSP去javaBean获取数据，然后返回给客户端



## JAVA知识

### Java内存模型

处理器的弱内存模型（内存屏障）：

执行一些特殊指令，将CPU缓存数据同步到内存中，

当程序需要读取某个变量值时，CPU中缓存的数据无效，然后需要去内存中获取

禁止重排序：

程序重排序可发生在编译器、运行、JIT时。下面有一个重排的例子

Class Reordering {

    int x = 0, y = 0;

    public void writer() {

x = 1;

     y = 2;

    }

public void reader() {

int r1 = y;

int r2 = x;

}

}

}

假如编译器对writer()中将y=2;放在了x=1之前

当有两个线程A、B, 其中A去调用writer()，已经执行完了y=2；B调用reader()，读取到了y=2，然而B再读取x的时候，并不一定x就等于1，因为这时候x=1发生在y=2之后。

##### synchronized

synchronized (this) {

}

当一个线程退出同步语句块时，线程释放锁对象（这里是this），它的作用是把CPU缓存的数据同步到主内存当中。

当另一个线程获取这个锁时，它在作用是使CPU缓存失效，从而使变量从主内存中重新加载。

##### volatile

对某个变量写入后，会被刷新到主内存中

读取某个变量时，会从主内存读取

此外，volatile禁止重排序,x=42一定发生在v=true之前

class VolatileExample {

 int x = 0;

 volatile boolean v = false;

 public void writer() {

   x = 42;

   v = true;

 }

 public void reader() {

   if (v == true) {

     //uses x - guaranteed to see 42.

   }

 }

}

### JVM内存模型

