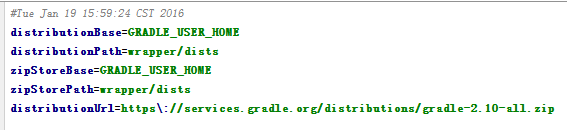
这个问题几乎每个刚使用[Android](http://lib.csdn.net/base/android) studio的同学都会碰到过，网上有各式各样的方法，有的说使用本地gradle，我试过多次，每次启动Android studio时还是会检查更新，所以根本上解决的办法就是阻止它更新。

其实道理很简单，找到工程里面自动更新请求的地址，把这个地址替换下就好了，不过这个简单的事儿还需要很多动作才能完成。

第一步，要自己下载gradle，下载后不必解压；

第二步，在Android studio中找到要修改的工程，工程名->gradle->wrapper->gradle-wrapper.properties，你会看到下图：



最后一行就是罪魁祸首，因为这个地址有时无法访问，导致卡住，我们把这个地址替换成可以访问的地址，就能彻底解决问题；

第三步，第二步之后，问题又来了，这个可以访问的地址该怎么设置呢，有人说把下载好的gradle传到网盘上，但我这样做之后，发现根本无法下载，不知道是不是只有我遇到了这个问题，如果你也遇到了，那就用我的办法，就是搭建一个本地服务器，然后把下载好的gradle放到服务器上；

第四步，这里就顺便说一下怎么搭建本地服务器吧，方法很简单，控制面板->程序->打开或关闭windows功能->Internet信息服务，如下图，把所有选项全部勾选



等待几分钟完成后，在浏览器输入http://localhost/，出现一个欢迎页面即搭建成功，若没有出现就多请求访问几次；

第五步，之后找到网站，一般在系统目录下面，我这里就是C:\inetpub\wwwroot，然后把下载好的gradle放到这个文件夹，然后终于就有了一个百分百可用的地址了；

第六步，回到第二步的配置文件，把最后一行地址替换为：http://localhost/gradle-2.10-all.zip，保存、重启，完事！

当然，完事之后可能会出现另一个错误，就是本地gradle的版本和目标版本不一致，这个就简单了，官网下载和目标版本一样的版本即可。

System类代表系统，系统级的很多属性和控制方法都放置在该类的内部。该类位于java.lang包。  
由于该类的构造方法是private的，所以无法创建该类的对象，也就是无法实例化该类。其内部的成员变量和成员方法都是static的，所以也可以很方便的进行调用。

**1、成员变量**

System类内部包含in、out和err三个成员变量，分别代表标准输入流(键盘输入)，标准输出流(显示器)和标准错误输出流(显示器)。  
例如：

System.out.println(“Test”);

该行代码的作用是将字符串”Test”输出到系统的标准输出设备上，也就是显示在屏幕上。  
后续在学习完IO相关的知识以后，可以使用System类中的成员方法改变标准输入流等对应的设备，例如可以将标准输出流输出的信息输出到文件内部，从而形成日志文件等。

**2、成员方法**

System类中提供了一些系统级的操作方法，这些方法实现的功能分别如下：

**a、arraycopy方法**

**public** **static** **void** **arraycopy**(Object src, **int** srcPos, Object dest, **int** destPos, **int** length)

该方法的作用是**数组拷贝**，也就是将一个数组中的内容复制到另外一个数组中的指定位置，由于该方法是native方法，所以性能上比使用循环高效。  
使用示例：

**int**[] a = {1,2,3,4};

**int**[] b = **new** **int**[5];

System.arraycopy(a,1,b,3,2);

该代码的作用是将数组a中，从下标为1开始，复制到数组b从下标3开始的位置，总共复制2个。也就是将a[1]复制给b[3]，将a[2]复制给b[4]，这样经过复制以后数组a中的值不发生变化，而数组b中的值将变成{0,0,0,2,3}。

**b、currentTimeMillis方法**

**public** **static** **long** **currentTimeMillis**()

该方法的作用是**返回当前的计算机时间**，时间的表达格式为当前计算机时间和GMT时间(格林威治时间)1970年1月1号0时0分0秒所差的**毫秒**数。例如：

**long** l = System.currentTimeMillis();

则获得的将是一个长整型的数字，该数字就是以差值表达的当前时间。 使用该方法获得的时间不够直观，但是却很方便时间的计算。例如，计算程序运行需要的时间则可以使用如下的代码：

**long** start = System. currentTimeMillis();

**for** (**int** i = 0;i < 100000000;i++) {

**int** a = 0;

}

**long** end = System. currentTimeMillis();

**long** time = end – start;

则这里变量time的值就代表该代码中间的for循环执行需要的毫秒数，使用这种方式可以测试不同算法的程序的执行效率高低，也可以用于后期线程控制时的精确延时实现。

**c、exit方法**

**public** **static** **void** **exit**(**int** status)

该方法的作用是退出程序。其中status的值为0代表正常退出，非零代表异常退出。使用该方法可以在图形界面编程中实现程序的退出功能等。

**d、gc方法**

**public** **static** **void** **gc**()

该方法的作用是请求系统进行垃圾回收。至于系统是否立刻回收，则取决于系统中垃圾回收算法的实现以及系统执行时的情况。

**e、getProperty方法**

**public** **static** String **getProperty**(String key)

该方法的作用是获得系统中属性名为key的属性对应的值。系统中常见的属性名以及属性的作用如下表所示。

| **属性名** | **属性说明** |
| --- | --- |
| java.version | Java运行时环境版本 |
| java.home | Java 安装目录 |
| [os.name](http://os.name/) | 操作系统的名称 |
| os.version | 操作系统的版本 |
| [user.name](http://user.name/) | 用户的账户名称 |
| user.home | 用户的主目录 |
| user.dir | 用户的当前工作目录 |

例如：

String osName = System.getProperty(“os.name”);

String user = System.getProperty(“user.name”);

System.out.println(“当前操作系统是：” + osName);

System.out.println(“当前用户是：” + user);

使用该方法可以获得很多系统级的参数以及对应的值。

# MVC模式深入理解

MVC是一种设计模式（Design pattern），也就是一种解决问题的方法和思路， 是上世纪80年代提出的，到现在已经颇有历史了。 MVC的意义在于指导开发者将数据与表现解耦，提高代码，特别是模型部分代码的复用性。

MVC不仅仅存在于Web设计中，在桌面程序开发中也是一种常见的方法。MVC的出现已经有一段历史了。 记得我最早了解到MVC的时候，是在Microsoft的Visual C++ 中的MFC中。 当时年少无知，以为是MFC中特有的东西。后来随之不断学习，才发现自己的天真。 所以说，学得越多，就越觉得自己无知。越觉得自己无知，就越懂得敬畏和谦逊。 从这个角度讲，同学们，最好不要看不起谦逊的人。

MVC是三个单词的缩写：Model,View,Controller。 MVC是一种设计模式，目前几乎所有的Web开发框架都建立在MVC模式之上。 当然，最近几年也出现了一些诸如MVP, MVVM之类的新的设计模式。 但从技术的成熟程度和使用的广泛程度来讲，MVC仍是主流。

Yii是一个Web框架，从Web开发的分工来讲，Yii的开发工作中，承担后端的内容多一些，毕竟主要就是PHP开发。 前端主要是在HTML、JavaScript、CSS上进行开发，然后通过Yii把前端的内容管起来，如通过Assets等。 这一章要讲的MVC，主要是针对后端的。 前端的MVC严格来讲不属于Yii的范畴，这里我们就不作过多介绍。 如果想了解前端的MVC，可以看看Backbone.js Angular.js等前端框架。

## MVC的三要素

MVC是**模型(Model)**、**视图(View)**、**控制器(Controller)** 3个单词的缩写。 下面我们从这3个方面来讲解MVC中的三个要素。

**Model**是指数据模型，是对客观事物的抽象。 如一篇博客文章，我们可能会以一个Post类来表示，那么，这个Post类就是数据对象。 同时，博客文章还有一些业务逻辑，如发布、回收、评论等，这一般表现为类的方法，这也是model的内容和范畴。 对于Model，主要是数据、业务逻辑和业务规则。相对而言，这是MVC中比较稳定的部分，一般成品后不会改变。 开发初期的最重要任务，主要也是实现Model的部分。这一部分写得好，后面就可以改得少，开发起来就快。

**View**是指视图，也就是呈现给用户的一个界面，是model的具体表现形式，也是收集用户输入的地方。 如你在某个博客上看到的某一篇文章，就是某个Post类的表现形式。 View的目的在于提供与用户交互的界面。换句话说，对于用户而言，只有View是可见的、可操作的。 事实上也是如此，你不会让用户看到Model，更不会让他直接操作Model。 你只会让用户看到你想让他看的内容。 这就是View要做的事，他往往是MVC中变化频繁的部分，也是客户经常要求改来改去的地方。 今天你可能会以一种形式来展示你的博文，明天可能就变成别的表现形式了。

**Contorller**指的是控制器，主要负责与model和view打交道。 换句话说，model和view之间一般不直接打交道，他们老死不相往来。view中不会对model作任何操作， model不会输出任何用于表现的东西，如HTML代码等。这俩甩手不干了，那总得有人来干吧，只能Controller上了。 Contorller用于决定使用哪些Model，对Model执行什么操作，为视图准备哪些数据，是MVC中沟通的桥梁。

对于MVC中三者的划分并没有十分明晰的定义和界线。MVC设计模式只是一种指导思想， 让你按照model, view, controller三个方面来描述你的应用，并通过三者的交互，使应用功能得以正常运转。

其中，View的部分比较明确，就是负责显示嘛。一切与显示界面无关的东西，都不应该出现在View里面。 因此，View中一般不会出现复杂的判断语句，不会出现复杂的运算过程。 对于PHP的Web应用而言，毫无疑问，HTML是View中的主要内容。这是关于View的几个原则：

* 负责显示界面，以HTML为主；
* 一般没有复杂的判断语句或运算过程，可以有简单的循环语句、格式化语句。 比如，一般博客首页的文章列表，就是一种循环结构；
* 从不调用Model的写方法。也就是说，View只从Model获取数据，而不直接改写Model，所以我们说他们老死不相往来。
* 一般没有任何准备数据的代码，如查询数据库、组合成一定格式的字符串等。 这些一般放在Controller里面，并以变量的形式传给视图。 也就是说，视图里面要用到的数据，都是拿来就能直接用的变量。

对于Model而言，最主要就是保存事物的信息，表征事物的行为和对他可以进行的操作。 比如，Post类必然有一个用于保存博客文章标题的title属性，必然有一个删除的操作，这都是Model的内容。 以下是关于Model的几个原则：

* 数据、行为、方法是Model的主要内容；
* 实际工作中，Model是MVC中代码量最大，逻辑最复杂的地方，因为关于应用的大量的业务逻辑也要在这里面表示。
* Model所提供的数据都是原始数据。也就是说，不带有任何表现层的代码。 比如，一般不会在输出的数据中添加HTML标签，这是View的工作。 但是Model可以提供有结构的数据，数组结构、队列结构、乃至其他Model等。 这个结构并非是表现层的格式，而是数据在内存中的表现。
* 与输出不同，Model的输入数据，可以是带有表现格式的数据。 如将一篇文章保存到Post里面，内容中必然包含各种HTML标签。 因此，Model一般要对输入数据作过滤、验证和规范化等预处理。 特别是对于需要保存进数据库的，一定要对所有的输入数据作预处理。 这些预处理，有的其实是业务逻辑。比如只有主编才可以删除文章，这一验证规则既也是业务逻辑，也是权限控制。 而有些预处理，则不属于业务逻辑，比如，文章标题前后的空格应去除。
* 注意与Controller区分开。Model是处理业务方面的逻辑，Controller只是简单的协调Model和View之间的关系， 只要是与业务有关的，就该放在Model里面。好的设计，应当是胖Model，瘦Controller。

对于Controller，主要是响应用户请求，决定使用什么视图，需要准备什么数据用来显示。 以下是有关Controller的设计原则：

* 用于处理用户请求。 因此，对于reqeust的访问代码应该放在Controller里面，比如 $\_GET $\_POST 等。 但仅限于获取用户请求数据，不应该对数据有任何操作或预处理，这些工作应该交由Models来完成。
* 调用Models的读方法，获取数据，直接传递给视图，供显示。 当涉及到多个Model时，有关的逻辑应当交给Model来完成。
* 调用Models的类方法，对Models进行写操作。
* 调用视图渲染函数等，形成对用户Reqeust的Response。

## Model设计参考

在MVC中，Model排第一，是有一定暗示的。一是Model是整个架构中，代码量最大，复用程度最高， 也是最体现程序员设计功力的地方。 二是View和Controller相对于Model而言，在实际开发中，复用程度不高，逻辑复杂程度较低。 可以说，Model设计得好，整个MVC就好，应用开发起就顺。

因此，这一节将以Model为核心，来讲MVC的设计。 实话说，MVC尽管提出了Model View Controller的划分思想，但到了具体实操中，并不是很好把握的。 下面介绍的设计参考，也仅仅是个人在实际项目中的一些体会和想法，仅作参考。 在具体设计中，可以后把握这么几点：

### Model应当集中整个应用的数据和业务逻辑

应用当中涉及到的所有业务对象都应尽可能抽像成Model。 如，博客系统当中，文章要抽象成Post，评论要抽象成Comment。 而相关的业务逻辑，如发布新文章可以用Post::create() ，删除评论可以用 Comment::delete()。 这样子整个应用就显得很清晰明了。

### 基础Model应当尽可能细化

在一个应用中，特别是对于大型复杂应用，Model间关系可能比较复杂。在构造应用时，特别是基础Model时， 要从足够小的粒度来设计。 此时，就要考虑采取继承、封装等措施了。 比如，一个博客文章Post，一般包含了若干标签，在页上一般写在作者、日期等Post字段的旁边。 从逻辑上来看，把标签作为Post的一个属性，是说得通的。 但是如果把标签作为一个属性像标题、正文等字段一样依附于Post。那么在有的功能上，实现起来是有难度的。 比如，客户要求，当一个Post含有标签 “yii, model” 时，可以点击 “yii” ， 然后系统列出所有具标签中含有 “yii” 的文章。

为了实现这个功能，正确的设计是单独将标签抽象成Tag。这样，Post和Tag是多对多的关系， 即一个Post有多个Tag，一个Tag也对应多个Post。这个多对多关系可以通过一张数据表 tbl\_post\_tag 来表示。 接下来，为Post增加Post::getTags()方法，并通过 tbl\_post\_tag 表来查询当前Post的所有标签。 同时，为Tag增加Tag::getPosts()方法，也通过 tbl\_post\_tag 表来查询当前Tag对应的文章。 这样，就具备了实现客户要求的新功能的基础。

因此，在Model设计上，要以尽量小的粒度进行设计。一般而言，粒度越小，复用的可能性就越高。

有的读者可能会问了，既然要求粒度尽可能地小，那么，Post是不是也应当再细化，把段落抽象为Model？ 是否有这个必要，看客户需求。一般情况确实没有这必要，如果这么做，那是不是再以句子为单位进行抽象？ 但如果客户要求这个博客系统的评论是针对段落进行的评论的， 要将评论显示在对应的段落旁边，甚至显示每个段落评论人次等功能。那么就需要把段落抽象成Model了。

### 分层次设计Model

从设计流程上，数据库结构设计与Model的设计是紧密相关的。先有数据库结构设计，后有Model设计。 在设计数据库结构的时候，也是在设计Model。 一般而言，最单元、粒度最小的Model就是根据每个数据库表所生成的Model，这往往是个Active Record。

比如标签的问题，在数据库存储过程中，Post和Tag是分开存的，而且这两个表的字段，没有冗余。 tbl\_post\_tag 表也只记录他们的ID，没有实质内容。

在获取数据渲染视图，向用户展现时，这两个Model及他们的字段，是完全够用，且没有冗余的。

那么，能不能说 Post 和 Tag 这两个Model是够用的呢？显然还不够。

当用户在创建文章、修改文章、审核文章时，需要采用一个表单来显示来收集用户输入。 其中，对于标签的采集，一般是一个长条的文本框，让用户一次性输入多个标签，并以 , 等进行分隔的。

但是，这个文本框没有一个字段与之进行对应。我们也没办法对这个字段的用户输入进行任何的验证、预处理。

因此，Post的功能是不够用的。不够用怎么办？那就加吧。但直接在 Post 里面加个public $tagString并不好。 毕竟只是在使用表单时，才会有这个问题，其他场合，这个字段是没用的。

这种情况下，一般使用继承:

public class **PostForm** extends **Post**

{

public $tagString;

... ...

}

这样，当控制器发现用户在创建、修改、审核文章时，可以使用 PostForm Model来渲染视图了， 而其他场合则仍使用Post。这样就在需要时，增加了一个 tagString 的字段用于收集用户输入的标签。

在具体设计过程中，由于Model本身就会包含很多代码，因此，要多使用这继承等手段，把代码组织好。

### 仔细为Model方法命名

由于Model的代码量比较大，又集中了大量的逻辑，因此，会在一个Model中有大量的方法。仍然以Post为例， 会涉及到创建、审核、发布、回收等流程，相关的方法比较多，在命名上要用心。 可能会涉及到的、名字又比较接近的方法就有：

* getPrevPost()，前一篇文章，用于导航
* getNextPost()，下一篇文章，用于导航
* getRelatedPosts($n = 10)，获取相关的N篇文章，用于相关文章推荐列表
* getPostsOfAuthor($n = 10)，获取同一作者的N篇相关文章，用于作者文章列表
* getLatestPosts($n = 10)，最新的N篇文章，静态方法，用于文章列表或RSS输出
* getHotestPosts($n = 10)，最热门的N篇文章，静态方法，用于热门文章列表
* getPublishPosts($n = -1)，获取已经发布的N篇文章，静态方法，用于文章列表
* getDraftPosts($n = -1)，获取未发布的N篇文章，静态方法，用于作者页面
* 这里只是一些获取其他Post的方法，命名比较合理，一看就知道意思。 而且全部写成getter的形式，可以使用读取属性的方式进行访问。

不单单是在Model方法的命名上要用心， 在变量名、类名、方法名等的命名上，也要养成习惯，形成规律。 不要图一时之快，胡乱起名。否则，出来混，迟早要还的。

## MVC与前后端的配合

从MVC的起源来讲，是从桌面应用的开发中发展起来的。从本质来讲，这是一种解决问题的思路和办法。 从实践来讲，这是一种久经考验的有效方式。但是如开头我们讲的，Yii更多的是侧重于后端。 对于Web应用而言，包含Yii在内的许多Web开发框架，都是没有办法知道用户的操作，如鼠标、键盘等操作的。 Web应用想要了解用户的操作，只能依靠用户发送Request。 而对于鼠标、键盘等的响应，只能依靠前端技术，如JavaScript等来实现。

再加上这几年来Web浏览器的功能日臻强大。因此，Web应用开发出现了一个新的发展苗头，就是功能从后端往前端转移。

在前端，通过JavaScript捕获用户操作，进行相应处理。 或是发送回后端获取响应后处理，如通过ajax请求后端数据，实现无刷新的局部页面更新，向用户进行反馈； 或直接在前端由浏览器进行处理，如使用backbone.js、Angular.js等前端框架的数据绑定功能等。 这些都使得Web应用表现得越来越像桌面应用。

后端MVC也在为前后端的发展而改变。 Controller的功能更多的变成了识别是ajax请求还是普通请求， 并根据请求的不同采取相应的视图渲染方式。对于普通请求，正常渲染视图，输出HTML。 对于ajax请求，则返回局部渲染视图，输出HTML片段。有的甚至输出XML或者JSON。 唯一在大潮流中，巍然不动的，还是我们的大Model。