**高级模型介绍**

尽管简单的模型和方块状态都很好，但它们不是动态的。例如，Forge的万能桶可以装载所有种类的可能是被Mod添加的流体。它必须动态地从基础的桶模型和液体把模型连接到一起。那是怎么做到的？点击IModel。

为了理解这是怎么运作的，让我们穿过模型系统的内部。通过这个小节，你可能将会不得不参考这个来抓住一个发生了什么的清晰的理解。这反过来也是正确的。你可能不会理解发生在这的一切事情，但是当你穿过这个部分你应该能够关于它抓住越来越多直到一切清楚了。

**重要**

如果这是您第一次阅读全篇请不要跳过*任何东西*！**按照顺序**读所有东西为了建立一个综合的理解是*势在必行的*！同样的，如果这是你第一次阅读，不要点击远离这节的链接。

1. 一套的ModelResourceLocation被标记为模型来通过ModelLoader被加载。
   * 对于物品，为了加载它们的模型必须手动地通过 ModelLoader.registerItemVariants被标记。(ModelLoader.setCustomModelResourceLocation做了这件事。)
   * 对于方块，它们的statemappers 提供了一个Map<IBlockState, ModelResourceLocation>。所有的方块都被迭代，并且这个map的值被标记来被加载。
2. [IModel](https://mcforge.readthedocs.io/en/latest/models/advanced/imodel/)s从每个ModelResourceLocation中被加载并且缓存在一个Map<ModelResourceLocation, IModel>里。
   * 一个IModel是从唯一的接收它的[ICustomModelLoader](https://mcforge.readthedocs.io/en/latest/models/advanced/icustommodelloader/) 中被加载的。(多个加载器试图加载一个模型将会导致一个LoaderException.)如果没寻找到并且这个ResourceLocation事实上是一个ModelResourceLocation (就是说，这不是一个普通的模型；它实际上是一个方块状态种类)，它会去方块状态加载器(VariantLoader)。否则这个模型就是一个普通的原版JSON模型并且以原版方式被加载(VanillaLoader)。
   * 一个原版的JSON模型(models/item/\*.json 或者 models/block/\*.json), 在加载时, 是一个 ModelBlock(是的，即使是物品). 这是一个原版的与IModel没有任何关系的类。为了改正这个，它被包装成一个*实现*了IModel 的VanillaModelWrapper。
   * 一个原版/Forge的方块状态种类，当加载时，首先读取整个的它来自的方块状态JSON。这个JSON被反序列化为一个ModelBlockDefinition然后缓存到JSON的目录。一个种类定义的列表然后从ModelBlockDefinition中被提取并且放到一个WeightedRandomModel中。
   * 当加载一个原版JSON物品模型(models/item/\*.json)时，模型用种类inventory (例如，泥土方块物品模型是minecraft:dirt#inventory)从一个ModelResourceLocation 中被请求；从而导致模型被VariantLoader(因为它是一个ModelResourceLocation)加载。当VariantLoader 无法加载来自方块状态JSON的模型时，它回滚到VanillaLoader。
     + 这个的最重要的副作用是如果一个错误发生在 VariantLoader中，它也将会尝试加载模型经由VanillaLoader。如果这也失败了，然后这个导致的异常会产生*两个* 堆栈跟踪。第一个是VanillaLoader的堆栈，第二个是VariantLoader的堆栈。当调试模型错误时，确保正确的堆栈被分析是重要的。
   * 一个IModel 可以从一个ResourceLocation中被加载或是从通过调用ModelLoaderRegistry.getModel或异常处理方案之一的缓存中恢复。
3. 所有加载的模型的材质依赖都被加载并且缝合为地图集。这个地图集是一个巨大的包括了所有粘在一起的模型材质。当一个模型引用一个材质时，在渲染期间，一个额外的UV偏移被应用来匹配在地图集中的材质的位置。
4. 每一个模型都被烘焙通过调用model.bake(model.getDefaultState(), ...).这结果的 [IBakedModel](https://mcforge.readthedocs.io/en/latest/models/advanced/ibakedmodel/) 被缓存在一个Map<ModelResourceLocation, IBakedModel>里。
5. 这个map然后被存储在ModelManager里。ModelManager 在游戏中的实例被存储在Minecraft::modelManager中，是私有的也没有getter。
   * ModelManager可能会被需要，不使用反射或是access tranformation，通过Minecraft.getMinecraft().getRenderItem().getItemModelMesher().getModelManager()或者Minecraft.getMinecraft().getBlockRenderDispatcher().getBlockModelShapes().getModelManager()。与它们的名字相反，它们是相同的。
   * 某人可能会从缓存中(不读取并且/或烘焙模型，仅仅通过存在的缓存) 用ModelManager::getModel在一个ModelManager中请求一个IBakedModel。
6. 最终，一个IBakedModel会被渲染。这被调用IBakedModel::getQuads完成。结果是一个BakedQuads(quadrilaterals: polygons with 4 vertices)的列表。然后这些能被送往GPU为了渲染物品和方块。物品和方块在那里有一点分散，但是下面的是相对简单的。（译者注：这句话没明白）
   * 原版的物品有属性和覆盖。为了使其更容易， IBakedModel定义了返回一个ItemOverrideList的getOverrides。ItemOverrideList定义了携带原有模型，实体，世界，和物品堆栈的handleItemState来寻找最终的模型。覆盖在对这个模型的所有其他操作之前被应用，包括getQuads。因为IBlockState不能应用于物品，IBakedModel::getQuads 接收到null 作为它的状态参数当作为物品被渲染时。
   * 方块有方块状态，并且当一个方块的IBakedModel 正在被渲染时，这个IBlockState 直接被送入getQuads方法。在仅有模型的情况下，方块状态会有一个额外的属性集，以[unlisted properties](https://mcforge.readthedocs.io/en/latest/models/advanced/extended-blockstates/)著称。