一.单机游戏　：　　数据存在本地。

　　txt(配置文件) 数据库

二.弱联网游戏 ：　　在需求的情况下 连一下网 ， 不是全程联网 。

　http:　get　post　 json 数据格式

三.强联网游戏 ：时时刻刻 跟服务器连接 。

　　class<==>二进制　,pb. (Protobuffer)

1，　单机游戏

　　txt(配置文件) 数据库

2，　弱联网游戏

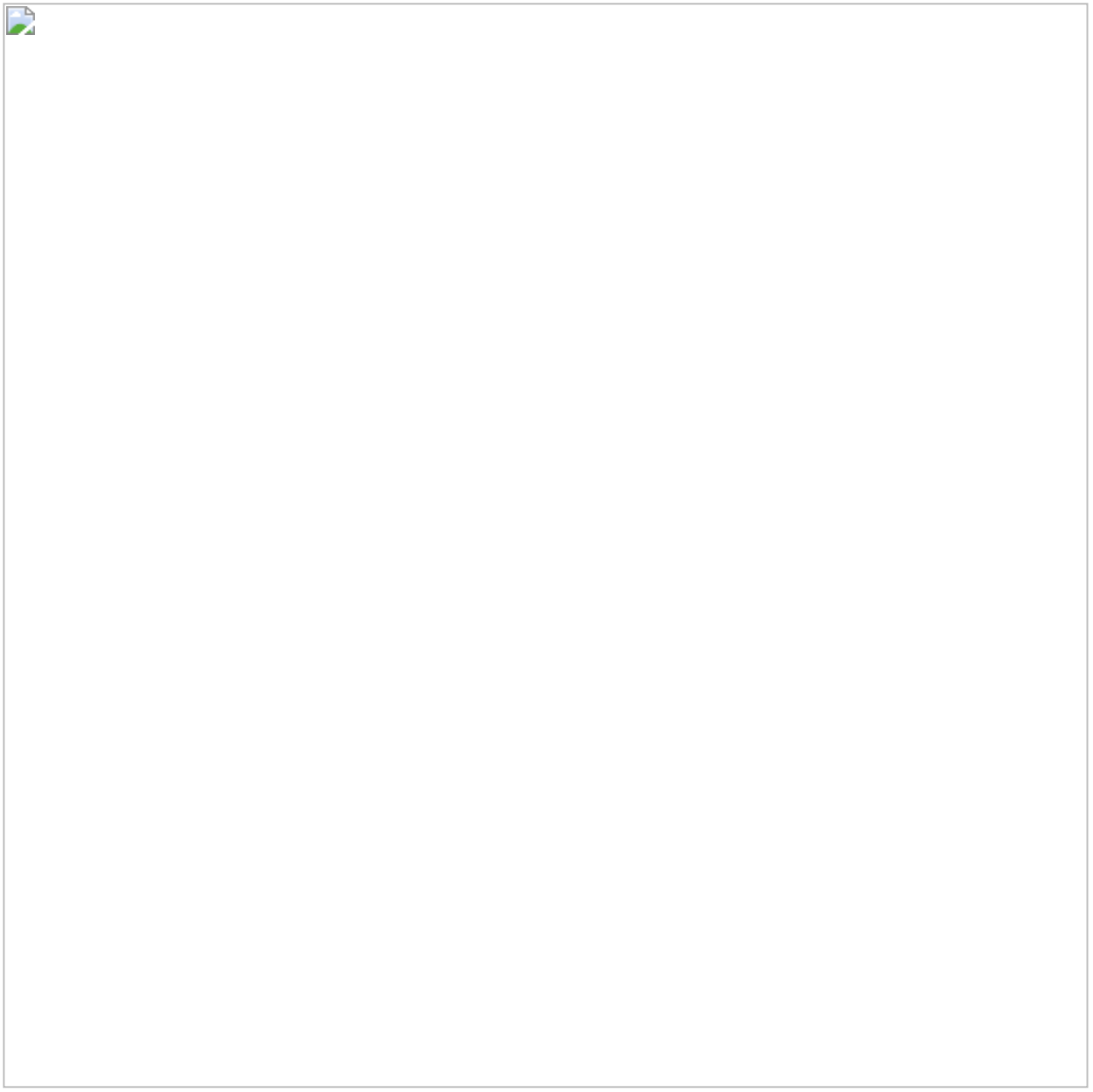
　http:　get　post　 json 数据格式

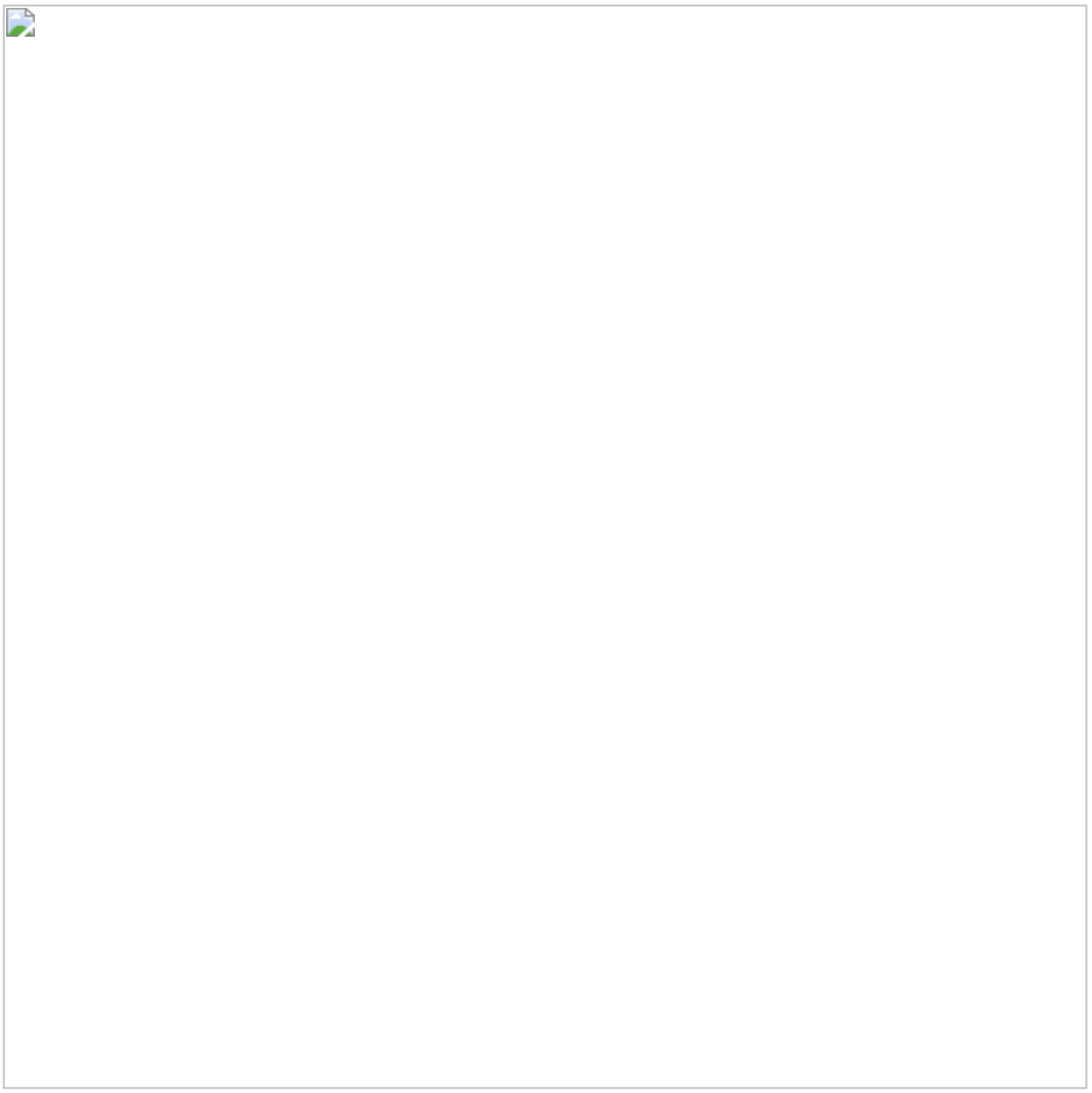
(1)Get 请求：

需要的时候 申请一下 服务器

浏览器 解析数据

- 参数在url 里面
- 最大长度是1024 字节



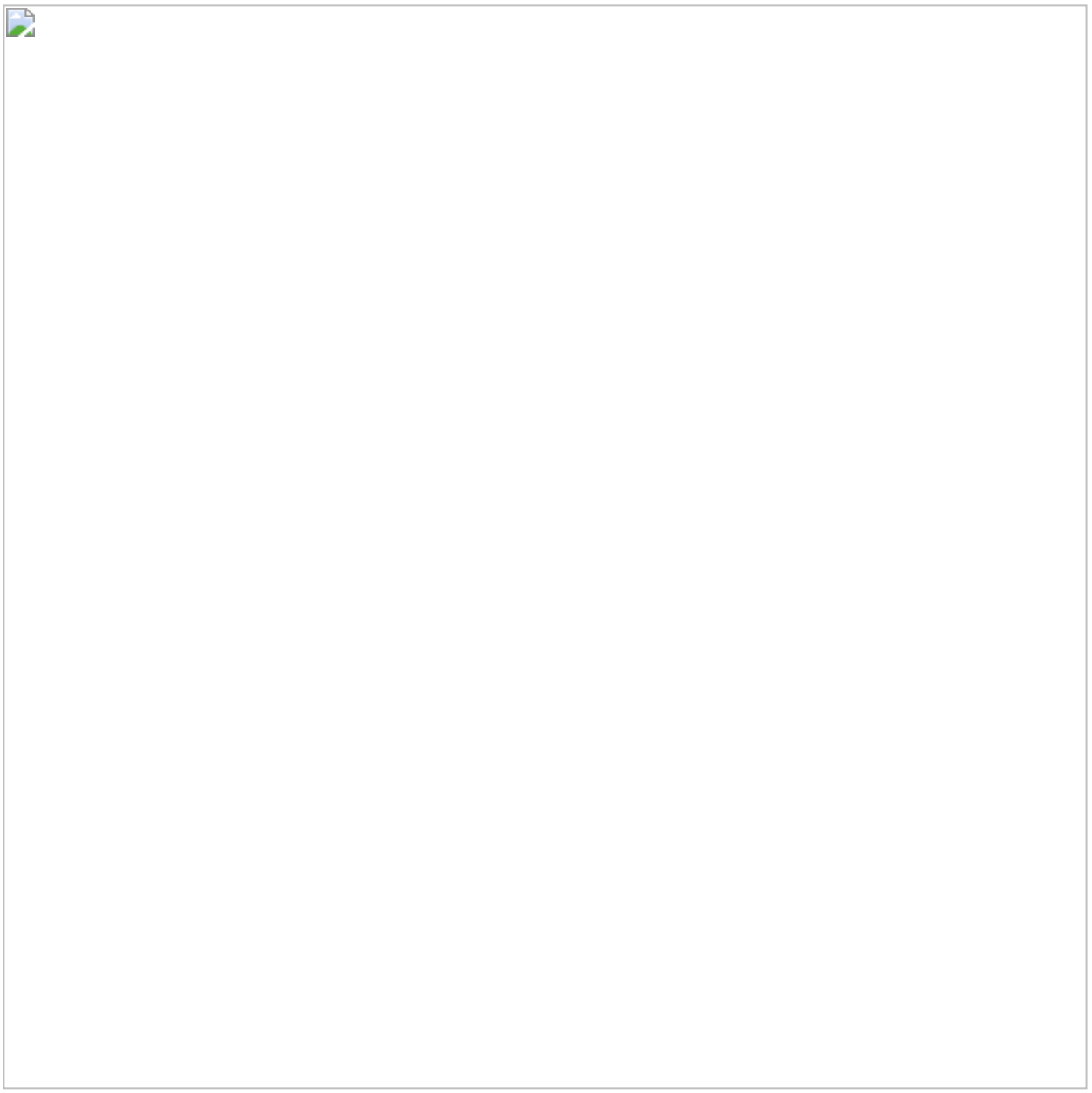


（2）Post：请求:

参数在表单里面 。 不在url 里面 。

申请大的数据 服务器 会申请一块大的内存。

- 1，参数不在url 里面 。在 body 里面 ---表单里面。
- 2，在服务器上申请一块大内存 。



（3）Json数据格式

Json 格式：

http://www.json.org/json-zh.html

json对象 ： { key: value , key： value ..... }

Json 数组： [{ key: value , key:value } , { key: value}]

常用的api 网站：https://www.juhe.cn/?

检验 json 格式对不对；https://www.sojson.com/

3， 强联网游戏

class<==>二进制 , pb.

（2）Protobffer:　　google

1，强联网游戏　尽量少的传递 数据。

2，同等量的信息数据　压缩更小。

C# 解析pb方式：

下载： proto Gen　这个工程 。

生成 一个 :protogen.exe

用 protogen　-i:test.proto　-o:myOut.cs　　命令生成 对应的pb.cs

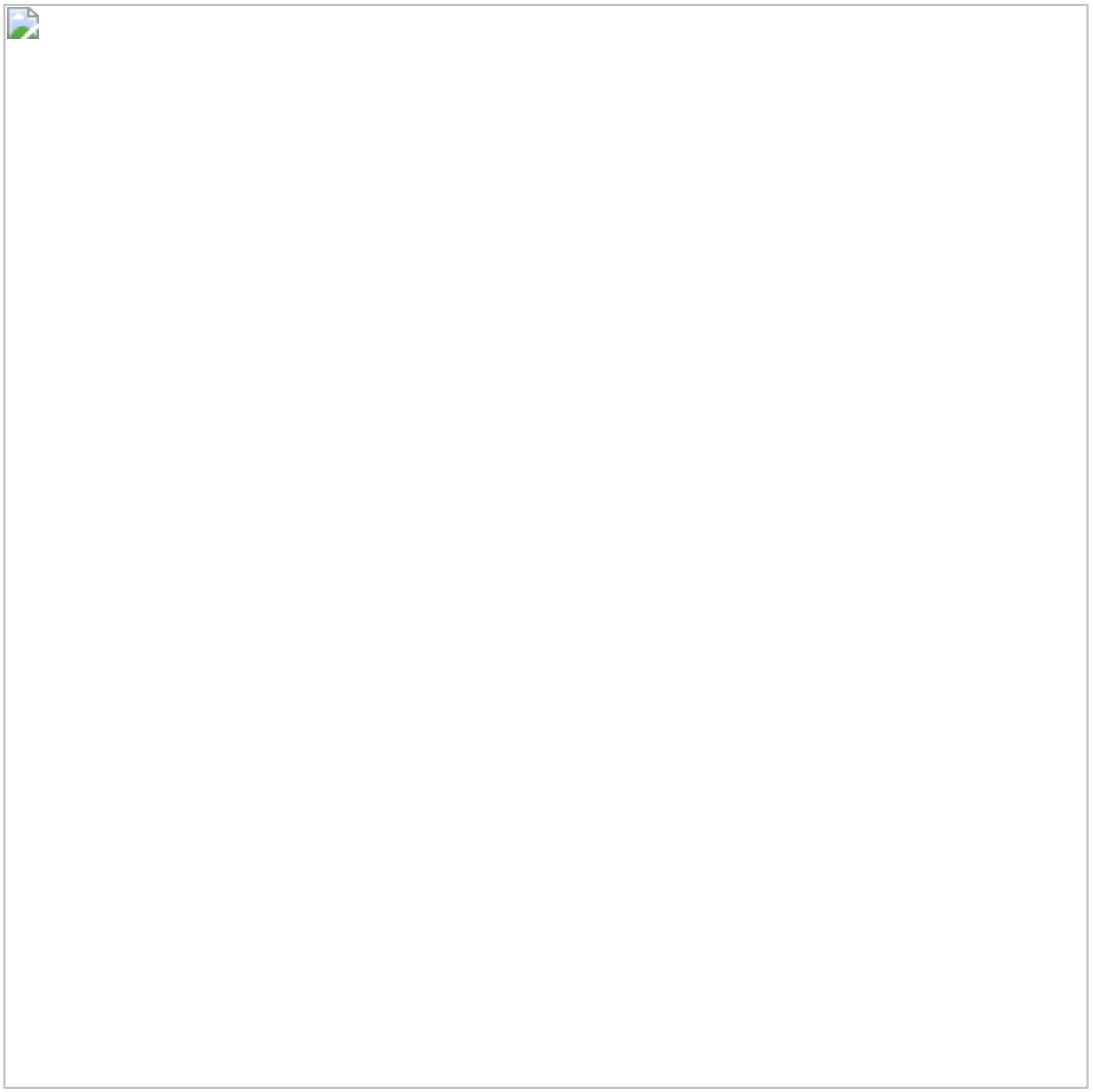
托管代码： 有托管堆 负责释放 自己的内存 。

非安全代码： 内存由自己控制　自己释放 。

设置工程：

1，　允许 非安全代码

2，　.net 2.0



将 用protogen　生成的cs 文件 拖入 工程　就可以使用了。

存储几个重要的目录 Path:

1，persistentPath . 当前unity3d 的工程目录。

可读可写

2，Application.dataPath ：表示当前工程目录下面的一个StreamingAssets目录。

当发布到真机上的时候，这个目录所有东西，会一起拷贝到真机上 。而且这个目录，在真机的时候只可读 不可写 。 WWW / webRequest . 读 ，

在开发的时候Editor 模式下 可读可写 。工程目录

3，Application.streamingAssets:在真机上，用文件系统 StreamReader FileStream可读可写 。 永久存在的一个目录 。

assets下面固定的文件夹，只能用webrequest 读。这个文件夹只能用UnityWebRequest去读,不可写。虽然这个文件夹 在PC 上可读可写 。

UnityWebRequest :

1， Get

2， Post

3， 申请图片。

4， 加载本地文件

UI 流程：

1， 拼界面

2， 从网络中 获取数据

3， 解析数据

4， 将解析的数据 更新到M 层

5， 将M 层的数据 刷新到UI 上。

界面开发流程：

1， 遵守MVC 思想 。

2， 获取需要用到的控件

3， 申请数据（从配置文件，从网络中， 从数据库中） 数据回来 更新M 层

4， 刷新UI 界面 。

1, 将 真机将要可读可写的东西A 先放入 streamingAssetsPath
2, 打包 发布到真机上的时候　　A 就连同 代码 资源 一起 放入到 真机上了
3, 将A 用UnityWebRequest　　读取出来 拷贝到 persistentDataPath
4，后面程序在运行的时候　就直接从 persistentDataPath　里面读取。
以上的　123 步骤 在第一次启动的时候去做。 只做一次 。
以上的 第四步　以后每次程序执行的过程 。

能够打包到真机的几个注意事项：
1，Resources　　动态加载目录
2，streamingAssetsPath　　文件目录
3，　场景中依赖的资源　或者　　Resources 里面依赖的资源
以上 3种资源 会被打入到安装包内 。 其它资源不会被打包进去。