小海螺核心算法报告

本篇报告将主要描述本项目（器乐练习辅助APP——小海螺）在工作原理、音频处理等方面的js核心算法的详细实现。由于能力所限，在描述内容中可能有错误之处，尽请原谅。

一、Core模块以及工作原理

对于算法的描述，先从核心处理模块Core开始。Core模块对应的是程序中的练习页面部分，其功能包括练习页上的更改乐谱、设定乐谱上的练习范围、调节节拍器频率，以及准备完成后的开始练习（即开启音频处理模块）。这四个功能在Core模块中对应四个不同的事件监听，如图1所示。



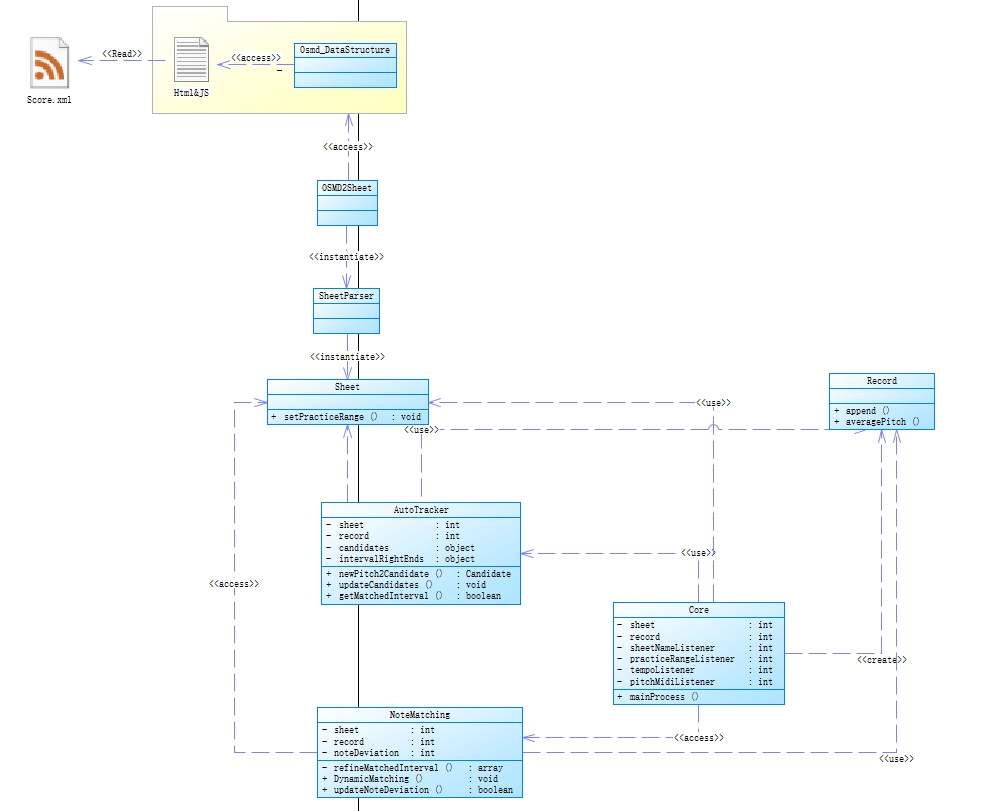
图1

Core模块在初始化时，会做一系列复杂的工作来准备程序的执行。首先在类中创建乐谱类sheet，音频处理类record，这两个对象可以被Core模块中所有方法使用。AutoTracker模块提供了从音频信号到具体某个音符的辨识功能，NoteMatcher模块中有更新音频的标准偏移量的方法，这两个功能经过初始化，可以为Core模块中的mainProcess所用。

在用户结束了所有准备工作，点击开始按钮之后，事件监听器会收到开始练习的信息，进入mainProcess。mainProcess的一次执行将会调用AutoTracker中的getMatchedInterval方法对一个音频信号进行辨识处理，如果辨识成功则会调用noteMatcher的updateNoteDeviation方法更新音频与标准音符的频率偏移量，否则将放弃此次处理，进入下一次循环。

二、乐谱的处理与读取

乐谱文件，通常保存了一首乐曲中面向演奏者的所有信息，包括节拍的格式、音符的长度和位置以及调式等等，通用的方法是将一首曲子的完整信息以xml的格式储存在文件中。我们使用了OSMD模块作为处理信息和乐谱显示的工具。



OSMD模块

乐谱显示

OSMD模块可以读取xml乐谱文件的各种信息并将其保存在内部的数据结构中。读取OSMD中我们需要的节拍、音符等信息，就可以得到一首完整的、包含各种要素的正确乐谱，作为和从用户演奏得来的音频信息进行比对的依据。

三、音频的处理

对音频的处理算法十分复杂。提取得到的原始频率数据存放在yinBuffer中。对于一次演奏者的演奏，采集到的信号通常如下图所示：

（1）对元素进行快速差分。

首先计算元素的幂项，然后进行快速傅里叶变换运算。在计算的过程中为了提高效率和性能，此处用到了第三方ooura库。现将计算所用数据存在缓存数组中，然后用ooura进行第一次快速傅里叶变换，结果存在缓存中。

（2）用一半的数据作为卷积核，进行第二次快速傅里叶变换。

（3）进行复乘法卷积运算。用ooura进行一次快速傅里叶逆变换。

（4）计算累计离差，将计算结果作为返回值。

通过以上三步，我们将音频信号进行了标准化，转化成可以在下一步骤中进行处理的数据格式。下面还需要对其进行具体的辨别，以得到与其相匹配的音符

**#E**

得到此结果之后，需要规定一定的阈值来判断每个音的起始点和终止点。这在absoluteThreshold函数中有所体现。该函数的策略是，取上升沿后的第一个极大指点作为单个音的起始点，以及下降沿前的最后一个极大值点作为单个音的终止点。

由于起始点实际上是一个数组的下标，而数组的下标只能为整数，因此若单单采取这种方式获取起始点的话，总会有比较大的误差。接下来应用了一种抛物线插值的方法，将起始点根据附近数据的差异大小进行调整，以得到更为准确的起始位置。

