# 汉字书写效果（字体可切换）

我们需要用到三个项目：

1、[makemeahanzi](https://github.com/skishore/makemeahanzi)     <https://github.com/skishore/makemeahanzi>

2、[hanzi-writer-data](https://github.com/chanind/hanzi-writer-data)    <https://github.com/chanind/hanzi-writer-data>

3、[hanzi-writer](https://github.com/chanind/hanzi-writer)    <https://github.com/chanind/hanzi-writer>

## makemeahanzi生成dictionary.txt和graphics.txt

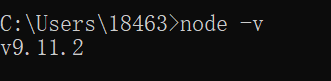
## hanzi-writer-data将dictionary.txt和graphics.txt转换成json文件



## [hanzi-writer](https://github.com/chanind/hanzi-writer)插件，将字体json文件导入，实现自定义字体的汉字书写效果

## 下面是具体步骤：

1. 首先需要安装node，传送门：<https://nodejs.org/en/download/>，安装完成后记得配置一下环境变量，cmd输入node -v 显示版本号即为安装成功。



1. 安装Meteor，安装环境参考Meteor官网 传送门：<https://www.meteor.com/install>
2. Linux命令：curl <https://install.meteor.com/> | sh
3. Windows下要先安装Chocolatey，传送门：<https://chocolatey.org/install>，安装有两种方式：

①：cmd安装：

@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass-Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))"&&SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"

②：powershell安装：

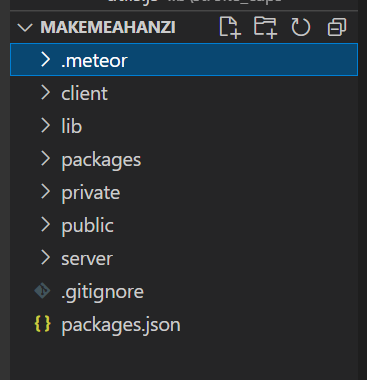
Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

PS：如果安装失败，可能是网络问题，需要梯子

安装完chocolatey之后，我们安装Meteor

choco install meteor

1. git上将meakmeahanzi项目clone下来之后，切换到tool分支

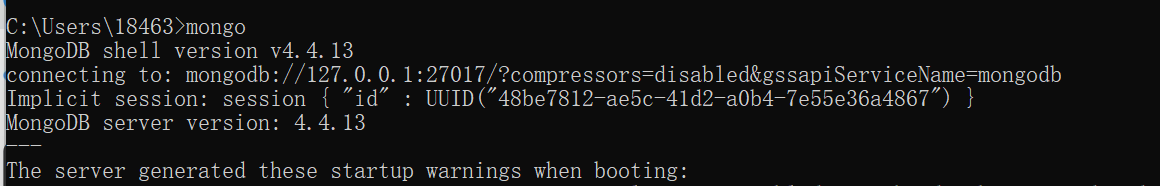
项目目录：

1. 安装mongodb数据库，传送门：

<https://www.runoob.com/mongodb/mongodb-window-install.html>

安装完成：

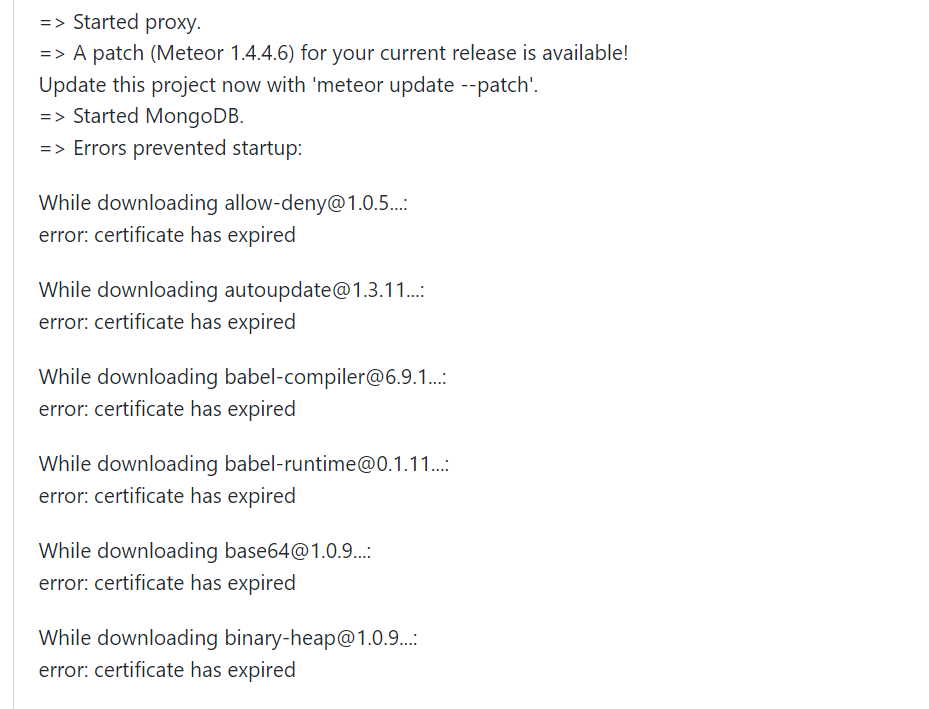
Win+r，然后输入cmd，打开命令提示窗，输入mongod



1. 运行项目：

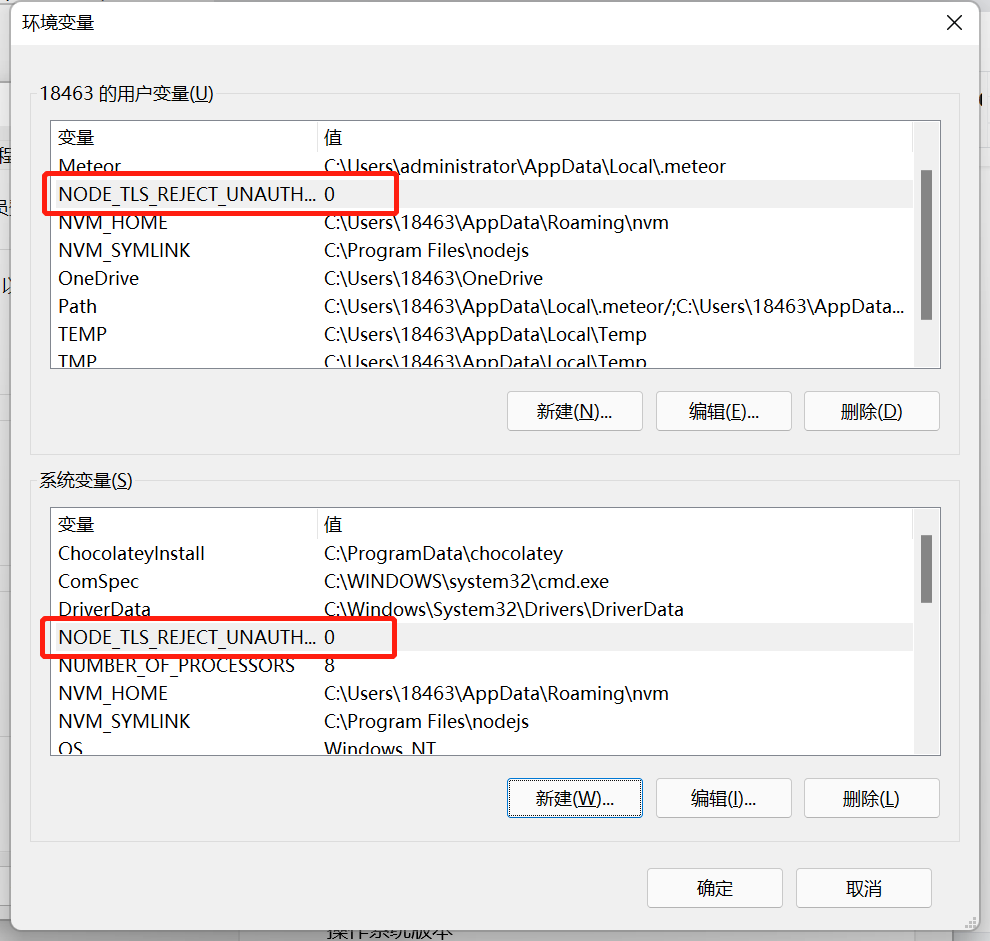
运行项目之前先把public文件夹拷贝到.meteor文件夹里面，Vscode打开项目，在根目录运行meteor run，等待项目跑起来，然后浏览器输入 [http://localhost:3000#](http://localhost:3000/)， 进入到server文件夹下输入meteor mongo可以打开mongo，mongo地址一般是localhost:3001端口

运行项目中有个坑点：由于meteor版本过低，为1.14.1.1，会出现证书不通过问题



解决方案：

1. 、mac系统：export NODE\_TLS\_REJECT\_UNAUTHORIZED=0; meteor run，就可以将项目正常运行
2. 、windows系统：需要在系统环境变量中设置NODE\_TLS\_REJECT\_UNAUTHORIZED=0，然后重启电脑



1. 项目运行起来之后的界面：



1. 使用项目写字

字体是可以自己导入的：

源码修改部分：

1)、client/editor.js对应的是编辑页面的逻辑，快捷键，操作步骤定义都在这个文件里。

2)、client/lib/path.js对应的是生成路径的逻辑，这里可以自己添加字体。添加字体的方法如下：

Template.path\_stage.helpers({

  alternative: () => Session.get('stages.path.alternative') || '?',

  options: () => [{font: 'arphic/gkai00mp.ttf', label: 'AR PL KaitiM GB'},

                  {font: 'arphic/UKaiCN.ttf', label: 'AR PL UKai'},

                  {font: 'arphic/simhei.ttf', label: 'simhei'},

                  {font: 'arphic/FZKTJW.TTF', label: 'FZKTJW'},

                  {font: 'arphic/FZLanTingYuanS-B-GB-TEST.ttf', label: 'FZLanTingYuanS-B-GB-TEST'},],

});

在options里面添加好字体后，将字体文件拷贝到.meteor/public/arphic/和.meteor/public/arphic/文件夹下即可

修改path.js使之兼容各种字体

本项目中用到的字体大小是1024的，如果一般字体没有这么大，比如256的，那么生成出来的字会很小，不能沾满整个画布，那么坐标也是无法使用的。我们可以修改path.js中的

// We avoid arrow functions in this map so that this is bound to the template.

Template.path\_stage.events({

  'blur .value': function(event) {

    const text = $(event.target).text();

    const value = text.length === 1 && text !== '?' ? text : undefined;

    if (value === stage.alternative) {

      $(event.target).text(value || '?');

    } else {

      stage.alternative = value;

      stage.forceRefresh();

    }

  },

  'click .option': function(event) {

    const label = this.label;

    const character = stage.character;

    assert(character.length === 1);

    Session.set('modal.text', `Loading ${label}...`);

    Session.set('modal.value', 0);

    opentype.load(this.font, (error, font) => {

      stage.alternative = undefined;

      if (error) {

        stage.onGetPath(`Error loading ${label}: ${error}`);

        return;

      }

      Session.set('modal.text', `Extracting ${character} from ${label}...`);

      Session.set('modal.value', 0.5);

      const index = font.charToGlyphIndex(character);

      const glyph = font.glyphs.get(index);

      if (glyph.unicode !== character.codePointAt(0)) {

        stage.onGetPath(`${character} is not present in ${label}.`);

        return;

      }

      const commands = font.getPath(character,0,0,1024).commands;

      // TODO(skishore): We may want a try/catch around this call.

      // const path = svg.convertCommandsToPath(glyph.path.commands);

      const path = svg.convertCommandsToPath(commands);

      stage.onGetPath(undefined, path);

    });

  },

});

红色部分即为修改的部分，这时候字体大小就会变成1024的大小，那么接下来会发现字体是反的，这时候需要修改/client/lib/external/opentype/0.4.10/opentype.js

Glyph.prototype.getPath = function(x, y, fontSize) {

    x = x !== undefined ? x : 0;

    y = y !== undefined ? y : 0;

    fontSize = fontSize !== undefined ? fontSize : 72;

    var scale = 1 / this.path.unitsPerEm \* fontSize;

    var p = new path.Path();

    var commands = this.path.commands;

    for (var i = 0; i < commands.length; i += 1) {

        var cmd = commands[i];

        if (cmd.type === 'M') {

            p.moveTo(x + (cmd.x \* scale), y + (cmd.y \* scale));

        } else if (cmd.type === 'L') {

            p.lineTo(x + (cmd.x \* scale), y + (cmd.y \* scale));

        } else if (cmd.type === 'Q') {

            p.quadraticCurveTo(x + (cmd.x1 \* scale), y + (cmd.y1 \* scale),

                               x + (cmd.x \* scale), y + (cmd.y \* scale));

        } else if (cmd.type === 'C') {

            p.curveTo(x + (cmd.x1 \* scale), y + (cmd.y1 \* scale),

                      x + (cmd.x2 \* scale), y + (cmd.y2 \* scale),

                      x + (cmd.x \* scale), y + (cmd.y \* scale));

        } else if (cmd.type === 'Z') {

            p.closePath();

        }

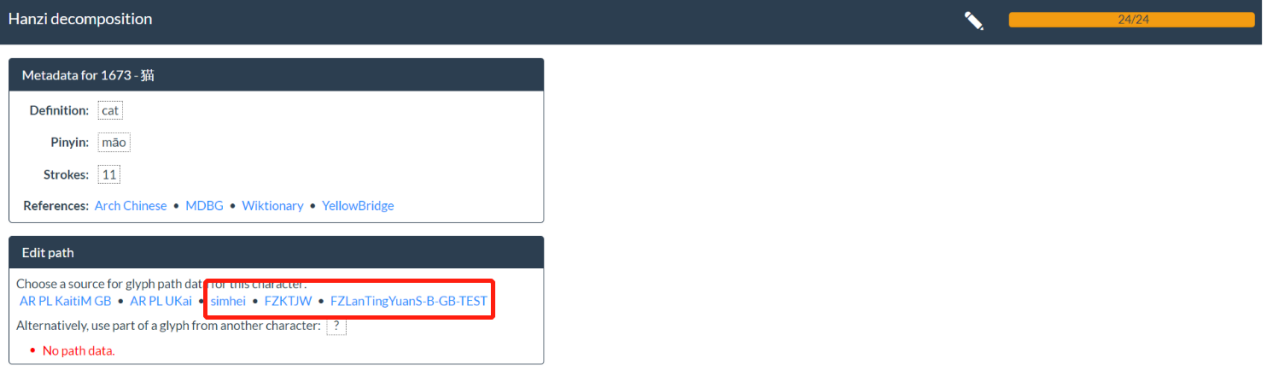
    }

    return p;

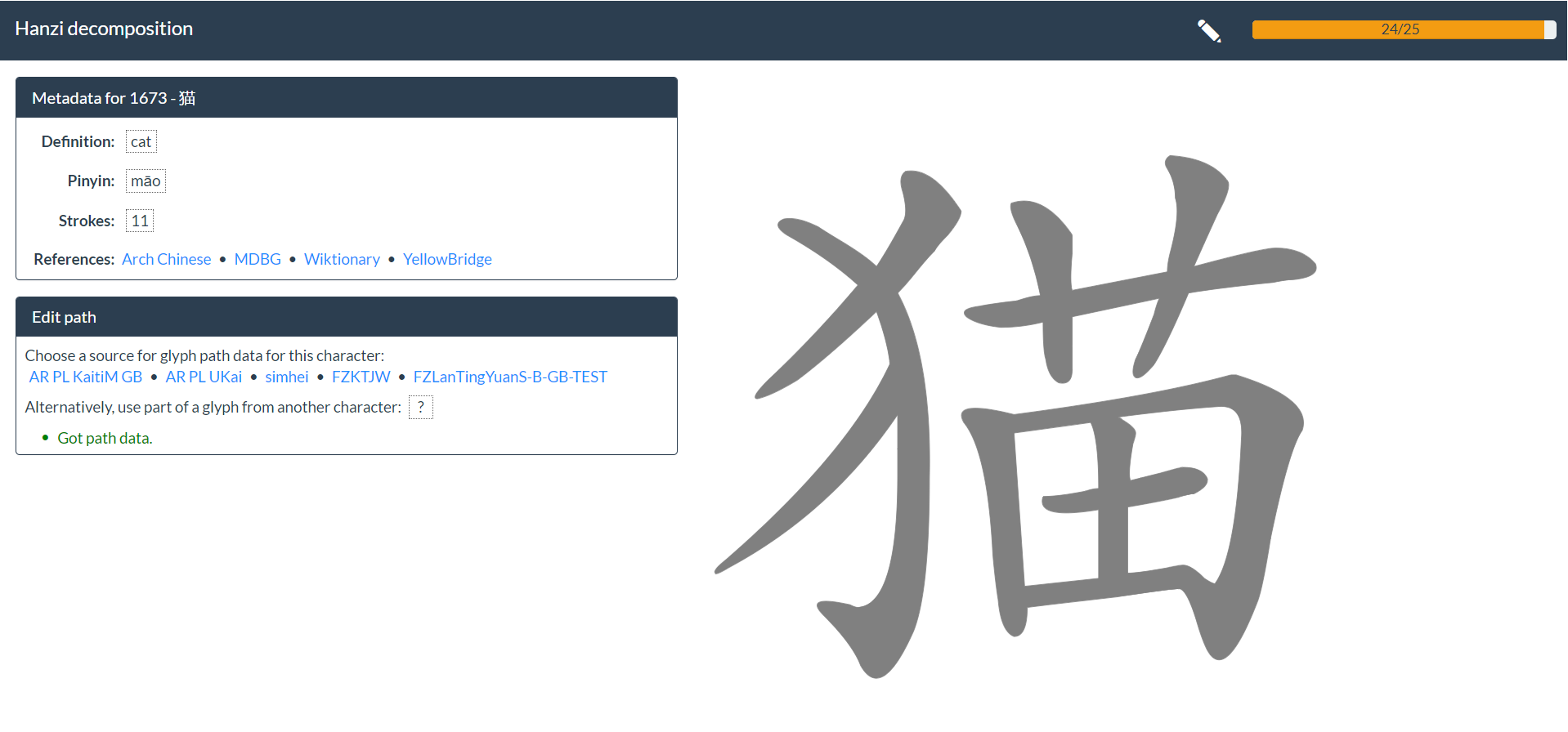
};

修改红色部分。这时候文字就会显示正常。

这样就会添加自己导入的几种字体



这时候点击一下字体，你可以生成对应字体的文字，下面是我点了FZKTJW的效果



使用方式：

快捷键a：显示上一个字

快捷键A：显示上一个未完成的字

快捷键q：显示上一个完成的字

快捷键d：显示下一个字

快捷键D：显示下一个未完成的字

快捷键e：显示下一个完成的字

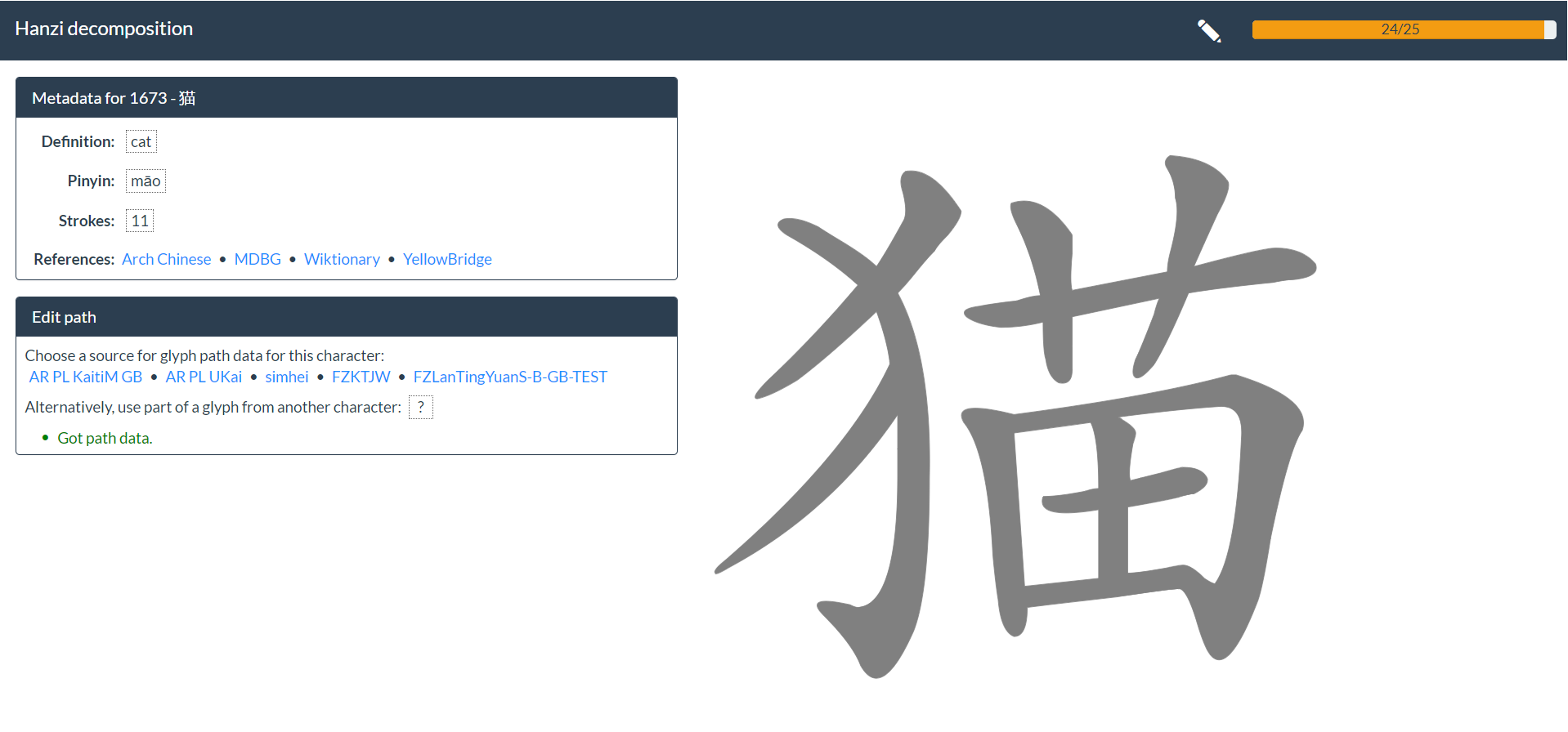
快捷键r：重置当前操作

快捷键s：下一步操作

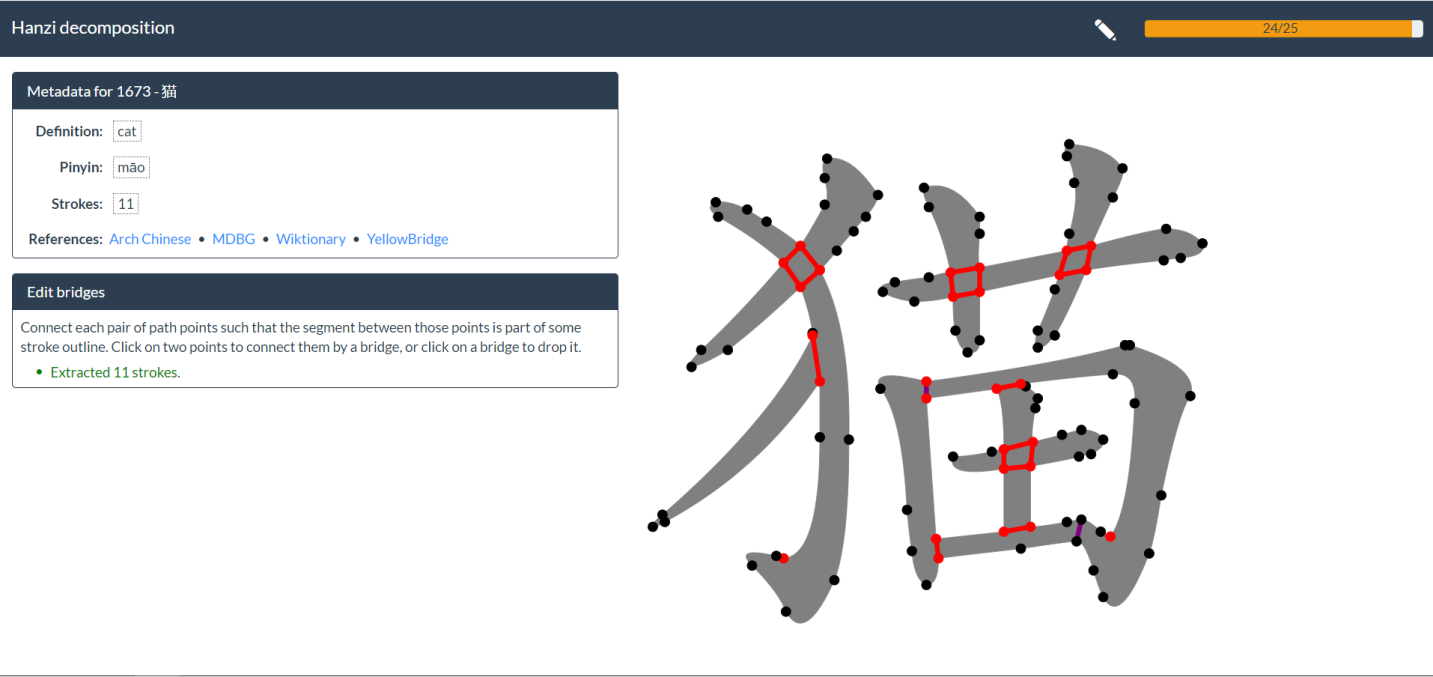
快捷键w：上一步操作

完成一个字需要六步操作，分别为：path, bridges, strokes, analysis, order, verified

path：路径



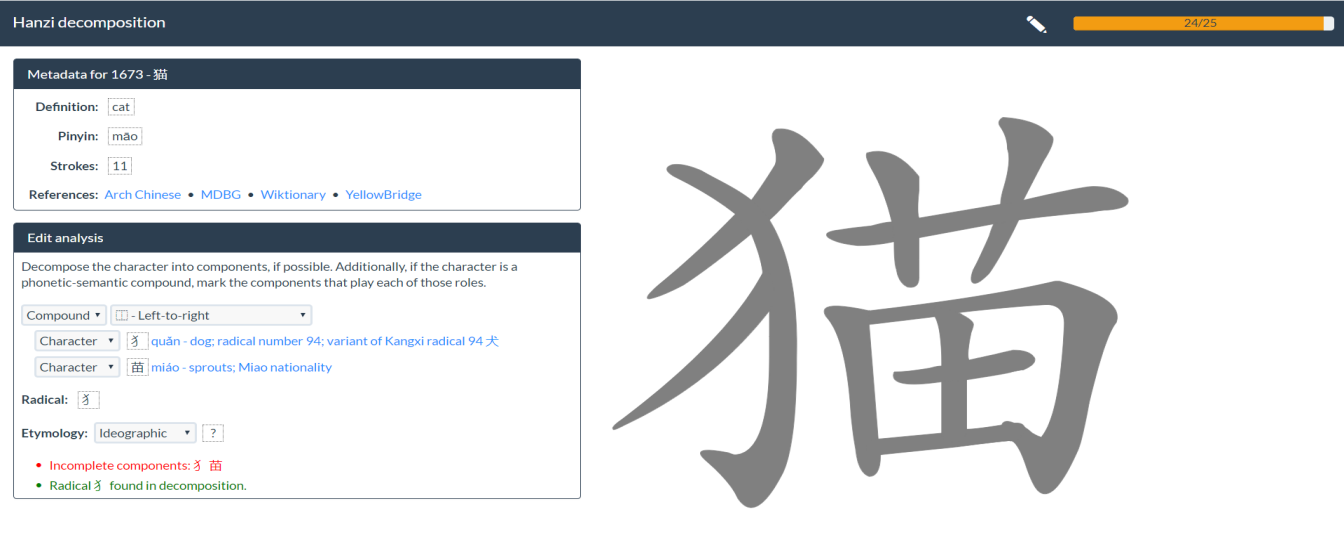
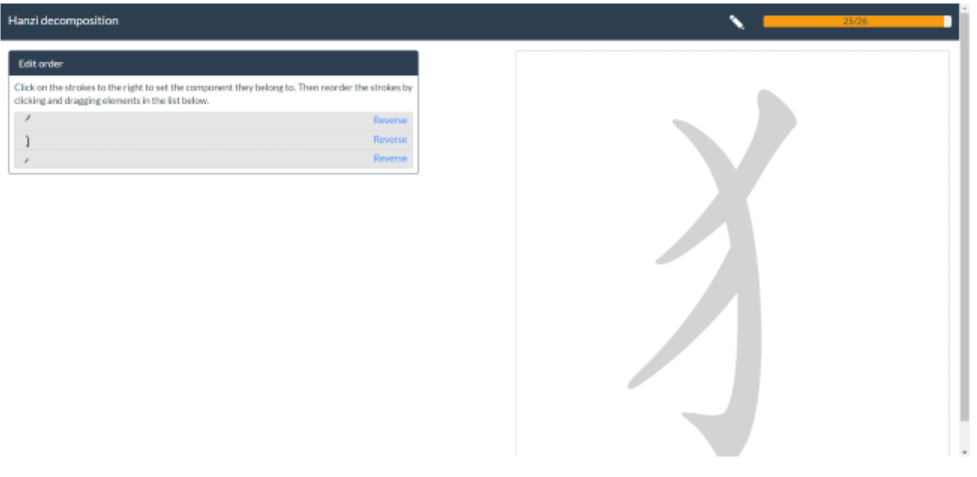
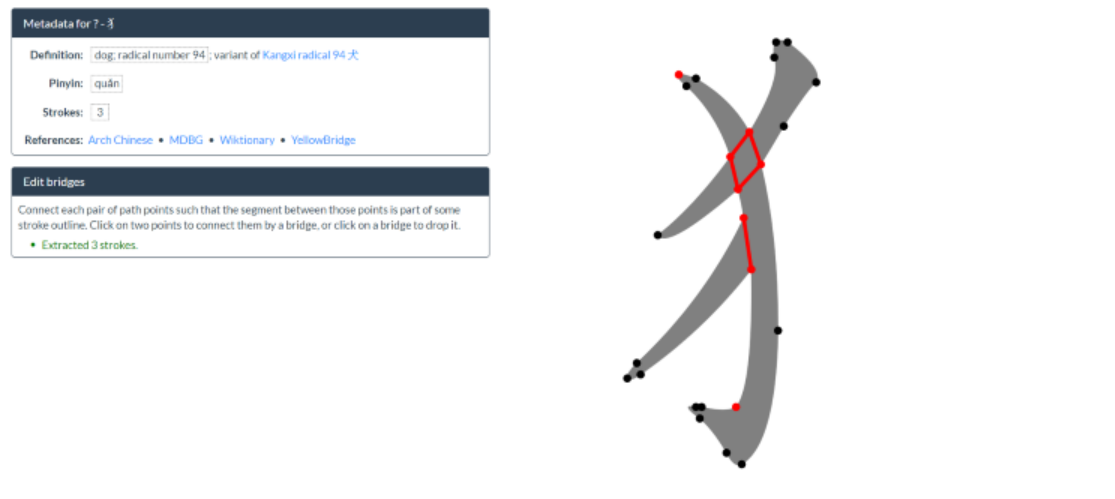
bridges：标记描点

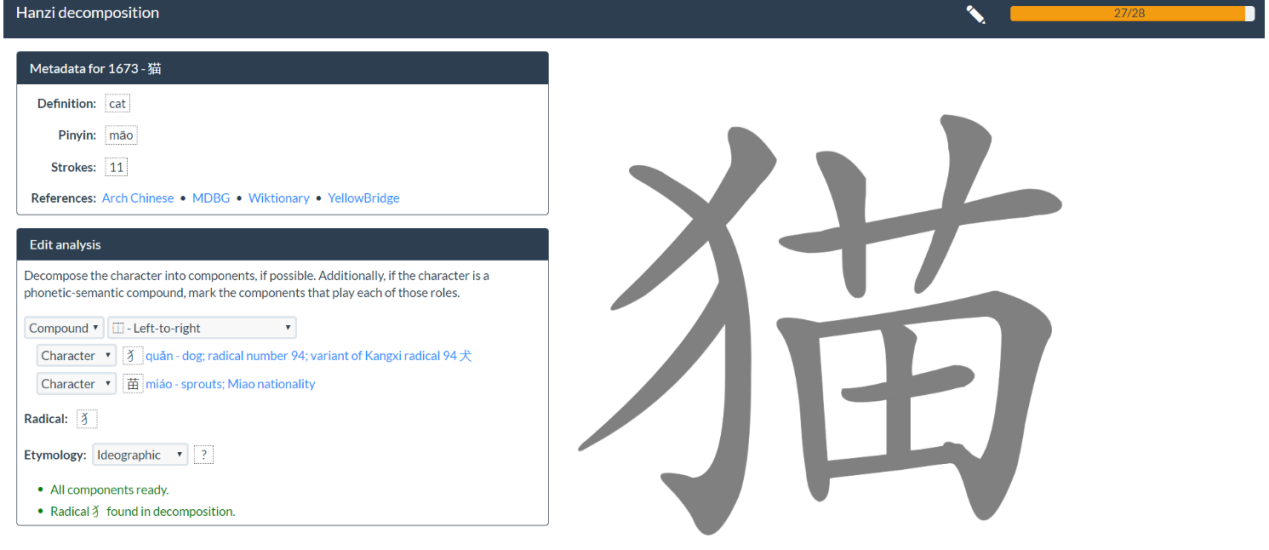
strokes：笔画

通过更改描点可以控制笔画

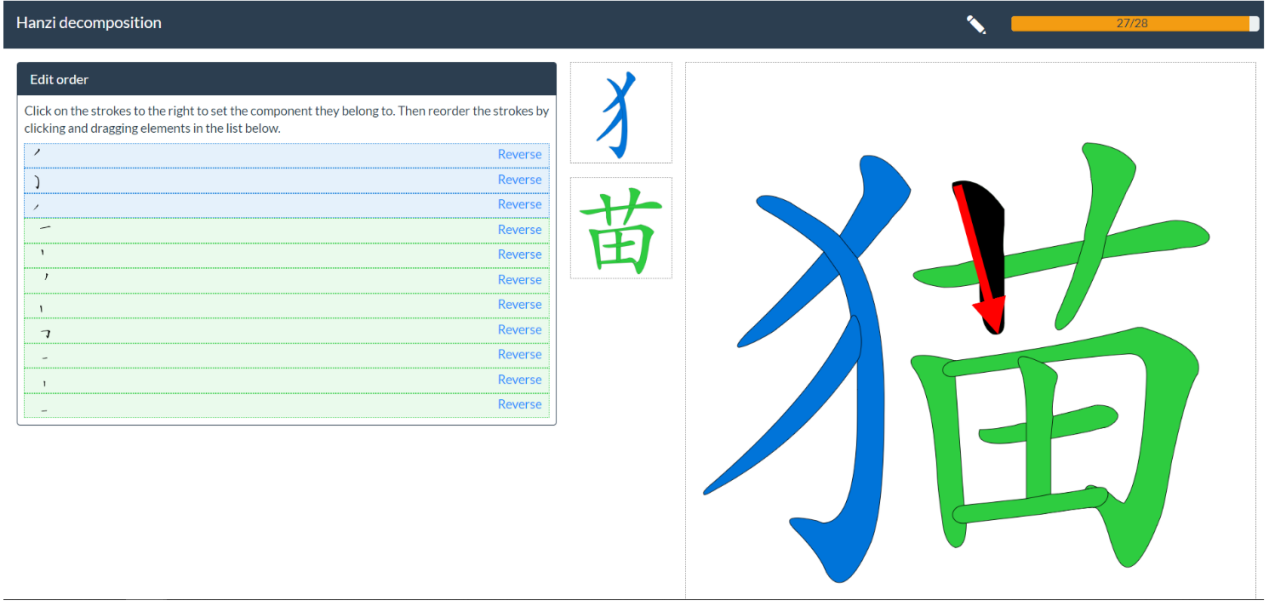
analysis：分析

分析字的结构，偏旁部首等待，如果结构没有完成，下一步会先去完成结构

对于“猫”这个字需要我们先完成“犭”和“苗”的构建，按s进入下一步

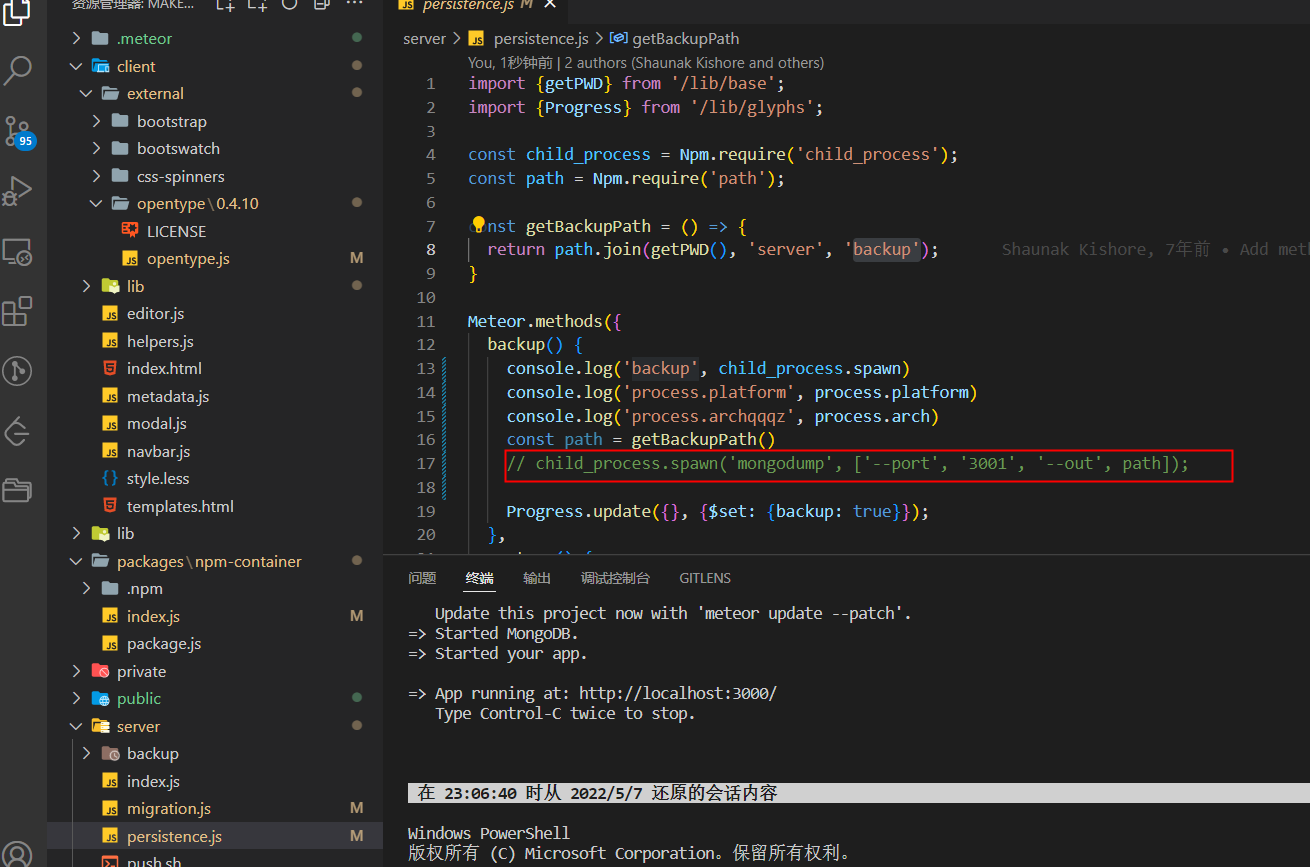
我们完成了“犭”的部分，接下来按D回到未完成的“猫”继续下一步按s进入下一步完成苗的构建此处跳过N部，总之构建完所有依赖的结构后按D回到未完成的“猫”继续搞按s进入下一步

order：顺序，笔顺

这里可以调整笔画的顺序（直接拖动）和笔顺Reverse是调整方向verified:检查并且完成

1. 导出：

导出时，需要将server/persistence.js中注释一行代码，如图：



## **导出路径文件和生成svg文件**

### **导出路径文件**

打开F12，切换到Console，然后输入

Meteor.call("backup");

进行一下备份，然后输入

Meteor.call('export');就可以在项目文件夹里面的.meteor文件夹下生成dictionary.txt和graphics.txt文件，稍后再讲这俩文件怎么用

### **生成svg文件**

F12，Console输入

Meteor.call('exportSVGs');

在项目文件夹.meteor文件夹下会生成.svg文件夹，里面就是字体的svg文件

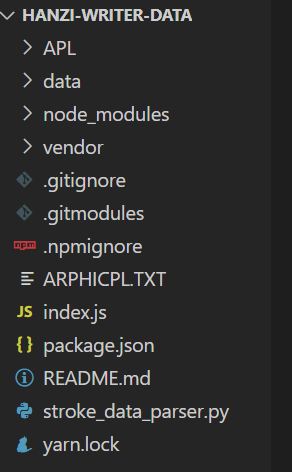
PS：如果生成的svg文件是空的，那么需要修改一下代码，将lib/animation.html 文件复制到private文件夹下并且覆盖原文件

1. [hanzi-writer-data](https://github.com/chanind/hanzi-writer-data)是python项目，需要先下载python，安装python3.x以上的版本

Python安装传送门：<https://www.python.org/getit/>

1. 此时可以我们进入下一步，使用meakmeahanzi得到的dictionary.js和graphics.txt

下载[hanzi-writer-data](https://github.com/chanind/hanzi-writer-data)，解压后的目录结构



打开stroke\_data\_parse.py文件可以看到

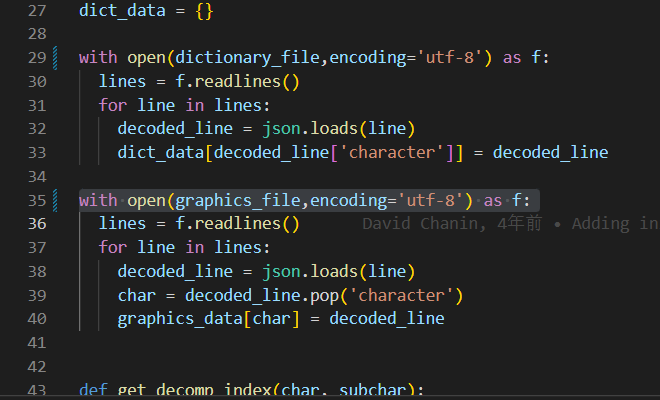
dictionary\_file = os.path.join(root, 'vendor/makemeahanzi/dictionary.txt')

graphics\_file = os.path.join(root, 'vendor/makemeahanzi/graphics.txt')

他是使用dictionary.txt和graphics.txt生成单个字独立的.json文件，以供[hanzi-writer](https://github.com/chanind/hanzi-writer-data)使用

我们将刚才生成的dictionary.txt和graphics.txt拷贝到vendor/makemeahanzi/目录下

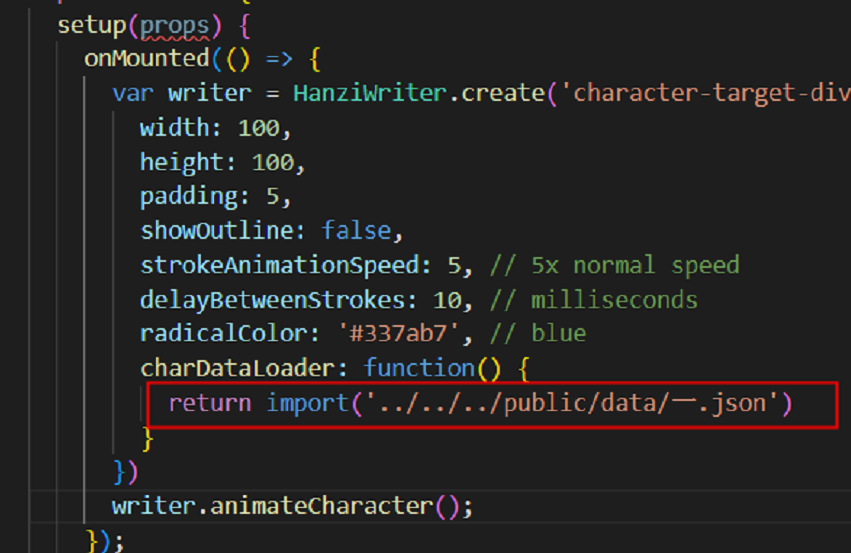
更改stroke\_data\_parser.py中的代码，设置utf-8编码



然后打开cmd输入python stroke\_data\_parse.py 会看到data文件夹下有生成好的json文件



1. 使用hanzi-writer，将刚才生成的json文件拷贝到hanzi-writer项目的demo文件夹，将里面的地址替换成刚才生成的all.json数据文件，然后打开后查找刚才我们自己构建的字



最终效果图：



关键技术

1.字形数据提取

字体是要符合trueType标准的ttf类型字体文件，

利用opentype.js解析ttf文件

1).对不同字体进行适配，转换尺寸，位置微调

2).提取所有字形glyph轮廓点contours和path数据

3).提取字形数据后用svg绘制出来

2.笔画拆取

要实现笔顺动画，就必须从源数据中拆取出所有笔画的轮廓数据，即笔画拆取。

由于笔画之间存在交叉重叠，若要实现笔画拆取，就需要将笔画交接处的凹点连通起来。这些处于交叉处特殊的拐角点称为corner，连通两个corner形成bridge，表明他们同属于一个笔画。

笔画拆取工作主要分为以下四步：

1. corner检测
2. corner匹配度计算
3. corner匹配搭建bridge
4. 依据bridge联通端点拆分笔画

实现：

EndPoint Parsing Corner检测

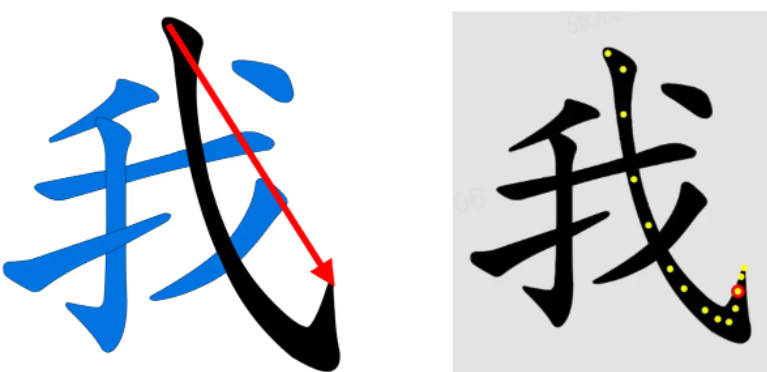
Corner Match Scoring 通过NN评分Corner匹配度

Corner Matching 通过匈牙利算法进行Corner匹配

Make Bridges 连通配对端点拆分笔画

3.笔画中点生成

上一步中实现了笔画的拆分，得到每个笔画的轮廓数据。依据轮廓数据可以进一步生成笔画的中点骨架。轮廓决定单个笔画的绘制范围，而中点则决定了绘制的顺序和方向。结合轮廓和中点数据，就可以实现单个笔画的绘制动画。



实现：

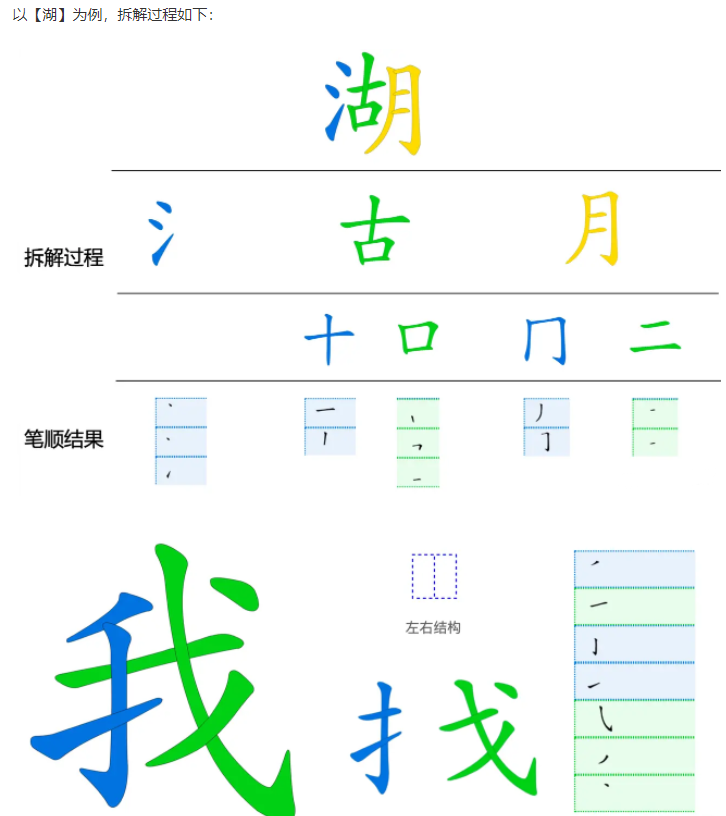
Polygon Approximation 端点加密

Polygon Voronoi 通过冯洛诺伊图（泰森多边形）生成中点

4.笔画顺序

根据不同汉字结构拆解笔顺，





拆解完笔顺后，需要将所有零散的中点数据，与当前所拥有的中点再做一遍匈牙利算法匹配，最终可得到一个相对正确的笔画顺序结果，生成json文件。

总结：

1. 将笔顺数据生成json文件存储在cdn上，文件大约4k左右大小；
2. 笔顺动画生产过程中用了比较多的推测对比算法，常规字体正确生成笔顺的概率高一些，但是也需要人工纠错。对于一些特殊风格字体算法偏差很大，需要人工介入手动生成；
3. 切换字体成本高，根据不同字体成本差别也很大；
4. 目前需要半人工半自动的方式生成字体笔顺数据，为降低人工成本需探索纠错算法，或者使用机器学习方式来提升生产效率。