/\* global api \*/

class enen\_Cambridge {

constructor(options) {

this.options = options;

this.maxexample = 2;

this.word = '';

}

async displayName() {

let locale = await api.locale();

if (locale.indexOf('CN') != -1) return '剑桥英汉双解(简体)';

if (locale.indexOf('TW') != -1) return '劍橋英漢雙解(簡體)';

return 'Cambridge EN->EN Dictionary';

}

setOptions(options) {

this.options = options;

this.maxexample = options.maxexample;

}

async findTerm(word) {

this.word = word;

let promises = [this.findCambridge(word), this.findYoudao(word)];

let results = await Promise.all(promises);

return [].concat(...results).filter(x => x);

}

async findCambridge(word) {

let notes = [];

if (!word) return notes; // return empty notes

function T(node) {

if (!node)

return '';

else

return node.innerText.trim();

}

let base = 'https://dictionary.cambridge.org/search/english /direct/?q=';

let url = base + encodeURIComponent(word);

let doc = '';

try {

let data = await api.fetch(url);

let parser = new DOMParser();

doc = parser.parseFromString(data, 'text/html');

} catch (err) {

return [];

}

let entries = doc.querySelectorAll('.pr .entry-body\_\_el') || [];

for (const entry of entries) {

let definitions = [];

let audios = [];

let expression = T(entry.querySelector('.headword'));

let reading = '';

let readings = entry.querySelectorAll('.pron .ipa');

if (readings) {

let reading\_uk = T(readings[0]);

let reading\_us = T(readings[1]);

reading = (reading\_uk || reading\_us) ? `UK[${reading\_uk}] US[${reading\_us}] ` : '';

}

let pos = T(entry.querySelector('.posgram'));

pos = pos ? `<span class='pos'>${pos}</span>` : '';

audios[0] = entry.querySelector(".uk.dpron-i source");

audios[0] = audios[0] ? 'https://dictionary.cambridge.org' + audios[0].getAttribute('src') : '';

//audios[0] = audios[0].replace('https', 'http');

audios[1] = entry.querySelector(".us.dpron-i source");

audios[1] = audios[1] ? 'https://dictionary.cambridge.org' + audios[1].getAttribute('src') : '';

//audios[1] = audios[1].replace('https', 'http');

let sensbodys = entry.querySelectorAll('.sense-body') || [];

for (const sensbody of sensbodys) {

let sensblocks = sensbody.childNodes || [];

for (const sensblock of sensblocks) {

let phrasehead = '';

let defblocks = [];

if (sensblock.classList && sensblock.classList.contains('phrase-block')) {

phrasehead = T(sensblock.querySelector('.phrase-title'));

phrasehead = phrasehead ? `<div class="phrasehead">${phrasehead}</div>` : '';

defblocks = sensblock.querySelectorAll('.def-block') || [];

}

if (sensblock.classList && sensblock.classList.contains('def-block')) {

defblocks = [sensblock];

}

if (defblocks.length <= 0) continue;

// make definition segement

for (const defblock of defblocks) {

let eng\_tran = T(defblock.querySelector('.ddef\_h .def'));

let chn\_tran = T(defblock.querySelector('.def-body .trans'));

if (!eng\_tran) continue;

let definition = '';

eng\_tran = `<span class='eng\_tran'>${eng\_tran.replace(RegExp(expression, 'gi'),`<b>${expression}</b>`)}</span>`;

chn\_tran = `<span class='chn\_tran'>${chn\_tran}</span>`;

let tran = `<span class='tran'>${eng\_tran}${chn\_tran}</span>`;

definition += phrasehead ? `${phrasehead}${tran}` : `${pos}${tran}`;

// make exmaple segement

let examps = defblock.querySelectorAll('.def-body .examp') || [];

if (examps.length > 0 && this.maxexample > 0) {

definition += '<ul class="sents">';

for (const [index, examp] of examps.entries()) {

if (index > this.maxexample - 1) break; // to control only 2 example sentence.

let eng\_examp = T(examp.querySelector('.eg'));

let chn\_examp = T(examp.querySelector('.trans'));

definition += `<li class='sent'><span class='eng\_sent'>${eng\_examp.replace(RegExp(expression, 'gi'),`<b>${expression}</b>`)}</span><span class='chn\_sent'>${chn\_examp}</span></li>`;

}

definition += '</ul>';

}

definition && definitions.push(definition);

}

}

}

let css = this.renderCSS();

notes.push({

css,

expression,

reading,

definitions,

audios

});

}

return notes;

}

async findYoudao(word) {

if (!word) return [];

let base = 'https://dict.youdao.com/w/';

let url = base + encodeURIComponent(word);

let doc = '';

try {

let data = await api.fetch(url);

let parser = new DOMParser();

doc = parser.parseFromString(data, 'text/html');

let youdao = getYoudao(doc); //Combine Youdao Concise English-Chinese Dictionary to the end.

let ydtrans = getYDTrans(doc); //Combine Youdao Translation (if any) to the end.

return [].concat(youdao, ydtrans);

} catch (err) {

return [];

}

function getYoudao(doc) {

let notes = [];

//get Youdao EC data: check data availability

let defNodes = doc.querySelectorAll('#phrsListTab .trans-container ul li');

if (!defNodes || !defNodes.length) return notes;

//get headword and phonetic

let expression = T(doc.querySelector('#phrsListTab .wordbook-js .keyword')); //headword

let reading = '';

let readings = doc.querySelectorAll('#phrsListTab .wordbook-js .pronounce');

if (readings) {

let reading\_uk = T(readings[0]);

let reading\_us = T(readings[1]);

reading = (reading\_uk || reading\_us) ? `${reading\_uk} ${reading\_us}` : '';

}

let audios = [];

audios[0] = `https://dict.youdao.com/dictvoice?audio=${encodeURIComponent(expression)}&type=1`;

audios[1] = `https://dict.youdao.com/dictvoice?audio=${encodeURIComponent(expression)}&type=2`;

let definition = '<ul class="ec">';

for (const defNode of defNodes){

let pos = '';

let def = T(defNode);

let match = /(^.+?\.)\s/gi.exec(def);

if (match && match.length > 1){

pos = match[1];

def = def.replace(pos, '');

}

pos = pos ? `<span class="pos simple">${pos}</span>`:'';

definition += `<li class="ec">${pos}<span class="ec\_chn">${def}</span></li>`;

}

definition += '</ul>';

let css = `

<style>

span.pos {text-transform:lowercase; font-size:0.9em; margin-right:5px; padding:2px 4px; color:white; background-color:#0d47a1; border-radius:3px;}

span.simple {background-color: #999!important}

ul.ec, li.ec {margin:0; padding:0;}

</style>`;

notes.push({

css,

expression,

reading,

definitions: [definition],

audios

});

return notes;

}

function getYDTrans(doc) {

let notes = [];

//get Youdao EC data: check data availability

let transNode = doc.querySelectorAll('#ydTrans .trans-container p')[1];

if (!transNode) return notes;

let definition = `${T(transNode)}`;

let css = `

<style>

.odh-expression {

font-size: 1em!important;

font-weight: normal!important;

}

</style>`;

notes.push({

css,

definitions: [definition],

});

return notes;

}

function T(node) {

if (!node)

return '';

else

return node.innerText.trim();

}

}

renderCSS() {

let css = `

<style>

div.phrasehead{margin: 2px 0;font-weight: bold;}

span.star {color: #FFBB00;}

span.pos {text-transform:lowercase; font-size:0.9em; margin-right:5px; padding:2px 4px; color:white; background-color:#0d47a1; border-radius:3px;}

span.tran {margin:0; padding:0;}

span.eng\_tran {margin-right:3px; padding:0;}

span.chn\_tran {color:#0d47a1;}

ul.sents {font-size:0.8em; list-style:square inside; margin:3px 0;padding:5px;background:rgba(13,71,161,0.1); border-radius:5px;}

li.sent {margin:0; padding:0;}

span.eng\_sent {margin-right:5px;}

span.chn\_sent {color:#0d47a1;}

</style>`;

return css;

}

}