```
import { LevelLoader } from "./LevelLoader";
import { Game } from "./views/Game";
import { LoadingViewRT } from "./views/loading/LoadingViewRT";
const { regClass, property } = Laya;
@regClass()
export class Boot extends Laya.Script {
    declare owner: Laya.Sprite;
    @property({ type: Laya.Prefab })
    private loadingPrefab: Laya.Prefab;
    @property({ type: LevelLoader })
    public levelLoader: LevelLoader;
    private _loadingNode: LoadingViewRT;
    //组件被激活后执行，此时所有节点和组件均已创建完毕，此方法只执行一次
    onAwake(): void {
        let node = this.loadingPrefab.create() as LoadingViewRT;
        this._loadingNode = node;
        node.value = 0;
        Laya.Scene.setLoadingPage(node);
        Laya.Scene.showLoadingPage();
        Game.ins.init(this);
    }
    setLoading(data: { desc?: string, value?: number }): void {
        data.desc != null && (this._loadingNode.desc = data.desc);
        data.value != null && (this._loadingNode.value = data.value);
    }
}
import { t, StateMachine } from "./core/FSM/StateMatchine";
export enum GameStates {
    init = "init",
    loading = "loading",
    loaded = "loaded",
    home = "home",
    level = "level",
    pause = "pause",
    win = "win",
}
export enum GameEvents {
    load = 100,
    loadComplete,
    enterHome,
    enterLevel,
    win,
    nextLevel,
    pause,
    resume,
    restartLevel,
    backHome,
}
export class GameFSM extends StateMachine<GameStates, GameEvents> {
    private readonly _id = `GameFSM_${Math.floor(Math.random() * 10000)}`;
    public onLoadHandler: Laya.Handler;
    public onLoadCompleteHandler: Laya.Handler;
    public onEnterHomeHandler: Laya.Handler;
    public onEnterLevelHandler: Laya.Handler;
    public onWinHandler: Laya.Handler;
    public onPauseHandler: Laya.Handler;
    public onNextLevelHandler: Laya.Handler;
    public onResumeHandler: Laya.Handler;
    public onRestartLevelHandler: Laya.Handler;
    public onBackHomeHandler: Laya.Handler;
    constructor(init = GameStates.init) {
        super(init);
        const s = GameStates;
        const e = GameEvents;
        this.addTransitions([
            t(s.init, e.load, s.loading, this.onLoad),
            t(s.loading, e.loadComplete, s.loaded, this.onLoadComplete),
            t(s.loaded, e.enterHome, s.home, this.onEnterHome),
            t(s.home, e.enterLevel, s.level, this.onEnterLevel),
            t(s.level, e.win, s.win, this.onWin),
            t(s.level, e.pause, s.pause, this.onPause),
            t(s.win, e.nextLevel, s.level, this.onNextLevel),
            t(s.win, e.restartLevel, s.level, this.onRestartLevel),
            t(s.win, e.backHome, s.home, this.onBackHome),
            t(s.pause, e.resume, s.level, this.onResume),
            t(s.pause, e.restartLevel, s.level, this.onRestartLevel),
            t(s.pause, e.backHome, s.home, this.onBackHome),
        ])
    }
    private logState(): void {
        this.logger.log(`${this._id} ${[GameStates[this.getState()]]}`);
    }
    private async onLoad(): Promise<void> {
        this.onLoadHandler && this.onLoadHandler.run();
    }
    private async onEnterHome(): Promise<void> {
        this.onEnterHomeHandler && this.onEnterHomeHandler.run();
    }
    private async onLoadComplete(): Promise<void> {
        this.onLoadCompleteHandler && this.onLoadCompleteHandler.run();
    }
    private async onEnterLevel(levelId: number): Promise<void> {
        this.onEnterLevelHandler && this.onEnterLevelHandler.runWith(levelId);
    }
    private async onWin(): Promise<void> {
        this.onWinHandler && this.onWinHandler.run();
    }
    private async onPause(): Promise<void> {
        this.onPauseHandler && this.onPauseHandler.run();
    }
    private async onNextLevel(levelId: number): Promise<void> {
        this.onNextLevelHandler && this.onNextLevelHandler.runWith(levelId);
    }
    private async onResume(): Promise<void> {
        this.onResumeHandler && this.onResumeHandler.run();
    }
    private async onRestartLevel(): Promise<void> {
        this.onRestartLevelHandler && this.onRestartLevelHandler.run();
    }
    private async onBackHome(): Promise<void> {
        this.onBackHomeHandler && this.onBackHomeHandler.run();
    }
}
import { SceneRegUtils } from "./core/UI/SceneRegUtils";
import { BackgroundRoot } from "./level/BackgroundRoot";
import { Level } from "./level/Level";
import { LevelLoadMask } from "./level/LevelLoadMask";
import { EViewLayer } from "./views/ViewConst";
import { LevelModel } from "./views/level/LevelModel";
/**
 * author: 陈秀齐
 * time: 2023/12/13 16:11:00
 * desc:
 */
const { regClass, property } = Laya;
@regClass()
export class LevelLoader extends Laya.Script {
    @property({ type: Laya.Prefab, tips: "关卡基础控件" })
    levelBasePrefab: Laya.Prefab;
    @property({ type: Laya.Prefab, tips: "视差滚动背景" })
    backgroundRootPrefab: Laya.Prefab;
    @property({ type: Laya.Prefab, tips: "场景过度动画节点" })
    loadMask: Laya.Prefab;
    get root(): Laya.Sprite {
        return this.owner.parent as Laya.Sprite;
    }
    createLoadMask(): LevelLoadMask {
        let node = this.loadMask.create();
        SceneRegUtils.tryAddChild(EViewLayer.UILoading, node as Laya.Sprite);
        return node.getComponent(LevelLoadMask);
    }
    createBackgroundRoot(): BackgroundRoot {
        let node = this.backgroundRootPrefab.create();
        this.root.addChild(node);
        return node.getComponent(BackgroundRoot);
    }
    loadLevel(levelId: number, backgroundRoot: BackgroundRoot): Level {
        let levelNode = this.levelBasePrefab.create();
        this.root.addChild(levelNode);
        let level = levelNode.getComponent(Level);
        level.init(levelId, backgroundRoot);
        return level;
    }
    unloadLevel(level: Level): void {
        LevelModel.ins.currId = null;
        level.owner.destroy();
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/14 19:19:50
 * desc:
 */
import { Boot } from "../Boot";
import { GameEvents, GameFSM, GameStates } from "../GameFSM";
import { ConfigPath } from "../const/ConfigPath";
import { SceneRegUtils } from "../core/UI/SceneRegUtils";
import { ViewMgr } from "../core/UI/ViewMgr";
import { ViewRegUtils } from "../core/UI/ViewRegUtils";
import { Singleton } from "../core/base/Singleton";
import { BackgroundRoot } from "../level/BackgroundRoot";
import { Level } from "../level/Level";
import { LevelLoadMask } from "../level/LevelLoadMask";
import { PromiseEx } from "../utils/PromiseEx";
import { ViewLayerZOrder, EViewKey, EViewLayer } from "./ViewConst";
import { ELevelConst } from "./level/LevelConst";
import { LevelModel } from "./level/LevelModel";
export class Game extends Singleton<Game>() {
    private _fsm: GameFSM;
    private _boot: Boot;
    private _level: Level;
    private _levelLoadMask: LevelLoadMask;
    private _backgroundRoot: BackgroundRoot;
    init(boot: Boot): void {
        this._boot = boot;
        this._fsm = new GameFSM();
        this._fsm.onLoadHandler = new Laya.Handler(this, this.onLoadHandler, null, false);
        this._fsm.onLoadCompleteHandler = new Laya.Handler(this, this.onLoadCompleteHandler, null, false);
        this._fsm.onEnterHomeHandler = new Laya.Handler(this, this.onEnterHomeHandler, null, false);
        this._fsm.onEnterLevelHandler = new Laya.Handler(this, this.onEnterLevelHandler, null, false);
        this._fsm.onWinHandler = new Laya.Handler(this, this.onWinHandler, null, false);
        this._fsm.onPauseHandler = new Laya.Handler(this, this.onPauseHandler, null, false);
        this._fsm.onNextLevelHandler = new Laya.Handler(this, this.onNextLevelHandler, null, false);
        this._fsm.onResumeHandler = new Laya.Handler(this, this.onResumeHandler, null, false);
        this._fsm.onRestartLevelHandler = new Laya.Handler(this, this.onRestartLevelHandler, null, false);
        this._fsm.onBackHomeHandler = new Laya.Handler(this, this.onBackHomeHandler, null, false);
        this._fsm.dispatch(GameEvents.load);
    }
    private async sequeueInit() {
        // 设置语言包
        await this.initLangPacks();
        this._boot.setLoading({ desc: "100003", value: 0.05 });
        await PromiseEx.delay(20);
        // 初始化场景层级
        this.buildScene();
        this._boot.setLoading({ desc: "100001", value: 0.10 });
        await PromiseEx.delay(20);
        // 初始化场景注册信息
        this.registerAllView();
        this._boot.setLoading({ desc: "100002", value: 0.15 });
        await PromiseEx.delay(20);
        // 设置加载 Common 资源
        this._boot.setLoading({ desc: "100004" });
        this.loadRes();
        Laya.SoundManager.setMusicVolume(0.5);
    }
    // 设置语言包
    private async initLangPacks(): Promise<any> {
        return Laya.loader.load(ConfigPath.JSON_Lang).then((result) => {
            Laya.Text.langPacks = result.data;
        })
    }
    // 初始化场景层级
    private buildScene(): void {
        let orders = ViewLayerZOrder;
        orders.forEach((item) => {
            const [layer, zOrder] = item;
            SceneRegUtils.add(layer, new Laya.Sprite(), zOrder);
        });
    }
    // 初始化场景注册信息
    private registerAllView(): void {
        const k = EViewKey, l = EViewLayer;
        ViewRegUtils.register(k.MainView, l.UI, { showMask: false, extraClick: false, enterAnim: false }, ConfigPath.LH_MainView);
        ViewRegUtils.register(k.SkinView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH_SkinView);
        ViewRegUtils.register(k.HelpView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH_Help);
        ViewRegUtils.register(k.HudView, l.UI, { showMask: false, extraClick: false, enterAnim: false }, ConfigPath.LH_Hud);
        ViewRegUtils.register(k.PauseView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH_PauseView);
        ViewRegUtils.register(k.WinView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH_WinView);
        ViewRegUtils.register(k.WinGoldView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH_WinGoldView);
    }
    private loadRes(): void {
        Laya.loader.on(Laya.Event.ERROR, this, this.onLoadError);
        Laya.loader.load(ConfigPath.EnterLoadList, null, Laya.Handler.create(this, this.onLoadProgress)).then(() => {
            this.onLoadCompleted();
        })
    }
    private openMainView(): void {
        this._backgroundRoot.setSkin(0);
        this._levelLoadMask.ungroup(Laya.Handler.create(this, () => {
            ViewMgr.ins.open(EViewKey.MainView);
            Laya.SoundManager.playMusic(ConfigPath.M_Main);
        }));
    }
    private onLoadCompleted(): void {
        this._boot.setLoading({ value: 0.95 });
        PromiseEx.delay(20).then(() => {
            this._boot.setLoading({ desc: "100005", value: 1.0 });
            Laya.Scene.hideLoadingPage(0);
            this._levelLoadMask = this._boot.levelLoader.createLoadMask();
            this._backgroundRoot = this._boot.levelLoader.createBackgroundRoot();
            this._backgroundRoot.enterAnim();
            Laya.timer.once(200, this, () => {
                this._fsm.dispatch(GameEvents.loadComplete);
            })
        });
    }
    private onLoadProgress(progress: number): void {
        console.log("progress=====", progress);
    }
    private onLoadError(error: string): void {
        console.log("error=====", error);
    }
    private onLoadHandler(): void {
        this.sequeueInit();
    }
    private onLoadCompleteHandler(): void {
        this._fsm.dispatch(GameEvents.enterHome);
    }
    private onEnterHomeHandler(): void {
        this.openMainView();
    }
    private onEnterLevelHandler(levelId: number): void {
        this._levelLoadMask.group(Laya.Handler.create(this, this.onEnterLevelGroupHandler, [levelId]));
    }
    private onEnterLevelGroupHandler(levelId: number): void {
        ViewMgr.ins.close(EViewKey.MainView);
        this._backgroundRoot.autoMove = false;
        this._backgroundRoot.setSkin(LevelModel.ins.skin);
        this._level = this._boot.levelLoader.loadLevel(levelId, this._backgroundRoot);
        ViewMgr.ins.open(EViewKey.HudView);
        this._levelLoadMask.ungroup();
        // Laya.SoundManager.playMusic(ConfigPath.M_Level);
    }
    private onWinHandler(): void {
        ViewMgr.ins.close(EViewKey.HudView);
        if (LevelModel.ins.isSecondLevel()) {
            ViewMgr.ins.open(EViewKey.WinGoldView);
        } else {
            ViewMgr.ins.open(EViewKey.WinView);
        }
    }
    private onPauseHandler(): void {
        ViewMgr.ins.close(EViewKey.HudView);
        ViewMgr.ins.open(EViewKey.PauseView);
    }
    private onNextLevelHandler(): void {
        this._levelLoadMask.group(Laya.Handler.create(this, this.onNextLevelGroupHandler));
    }
    private onNextLevelGroupHandler(): void {
        const levelId = ELevelConst.Level_10002;
        this._level.reEnterLevel(levelId);
        ViewMgr.ins.close(EViewKey.WinView);
        ViewMgr.ins.open(EViewKey.HudView);
        this._levelLoadMask.ungroup();
        // Laya.SoundManager.playMusic(ConfigPath.M_Level);
    }
    private onResumeHandler(): void {
        ViewMgr.ins.open(EViewKey.HudView);
        ViewMgr.ins.close(EViewKey.PauseView);
    }
    private onRestartLevelHandler(): void {
        this._levelLoadMask.group(Laya.Handler.create(this, this.onRestartLevelGroupHandler));
    }
    private onRestartLevelGroupHandler(): void {
        this._levelLoadMask.ungroup();
        ViewMgr.ins.close(EViewKey.PauseView);
        ViewMgr.ins.close(EViewKey.WinGoldView);
        this._level.restart();
        ViewMgr.ins.open(EViewKey.HudView);
        // Laya.SoundManager.playMusic(ConfigPath.M_Level);
    }
    private onBackHomeHandler(): void {
        this._levelLoadMask.group(Laya.Handler.create(this, this.onBackHomeGroupHandler));
    }
    private onBackHomeGroupHandler(): void {
        this._boot.levelLoader.unloadLevel(this._level);
        this._level = null;
        ViewMgr.ins.close(EViewKey.PauseView);
        ViewMgr.ins.close(EViewKey.WinGoldView);
        this.openMainView();
    }
    enterLevel(levelId: number): void {
        this._fsm.dispatch(GameEvents.enterLevel, levelId);
    }
    pause(): void {
        this._fsm.dispatch(GameEvents.pause, "param");
    }
    resume(): void {
        this._fsm.dispatch(GameEvents.resume);
    }
    restartLevel(): void {
        this._fsm.dispatch(GameEvents.restartLevel);
    }
    backHome(): void {
        this._fsm.dispatch(GameEvents.backHome);
    }
    nextLevel(): void {
        this._fsm.dispatch(GameEvents.nextLevel);
    }
    win(): void {
        this._fsm.dispatch(GameEvents.win);
    }
    isWin(): boolean {
        return this._fsm.getState() === GameStates.win;
    }
    scrollTo(p: number): void {
        this._level.scrollTo(p);
    }
}
export enum EViewKey {
    HudView = "HudView",
    PauseView = "PauseView",
    HelpView = "HelpView",
    MainView = "MainView",
    SkinView = "SkinView",
    WinView = "WinView",
    WinGoldView = "WinGoldView",
}
export enum EViewLayer {
    Bg = "bg",
    Battle = "battle",
    UI = "ui_full",
    UISystem = "ui_main",
    UIPopup = "ui_popup",
    UIMsg = "ui_msg",
    UIGuide = "ui_guide",
    UILoading = "ui_loading",
    UIAlert = "ui_alert",
}
export const ViewLayerZOrder: [EViewLayer, number][] = [
    /**Bg 层 */
    [EViewLayer.Bg, 10],
    /**fight 层 */
    [EViewLayer.Battle, 20],
    /**UI 层 */
    [EViewLayer.UI, 100],
    /**一级窗口 UI 层 */
    [EViewLayer.UISystem, 200],
    /**二级弹窗 UI 层 */
    [EViewLayer.UIPopup, 300],
    /**飘字信息 UI 层 */
    [EViewLayer.UIMsg, 400],
    /**引导层 */
    [EViewLayer.UIGuide, 500],
    /**loading */
    [EViewLayer.UILoading, 600],
    /**提示窗口层 */
    [EViewLayer.UIAlert, 700],
];
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/Help.lh
 */
export class HelpViewRTBase extends Laya.Box {
    public lblTitle!: Laya.Label;
    public lblDesc!: Laya.Label;
    public btnClose!: Laya.Button;
}
const { regClass } = Laya;
import { ViewMgr } from "../../core/UI/ViewMgr";
import { EViewKey } from "../ViewConst";
import { HelpViewRTBase } from "./HelpViewRT.generated";
@regClass()
export class HelpViewRT extends HelpViewRTBase {
    private onClickClose(): void {
        ViewMgr.ins.close(EViewKey.HelpView);
    }
    onAwake(): void {
        this.btnClose.on(Laya.Event.CLICK, this.onClickClose);
    }
}
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/HUD.lh
 */
export class HudViewRTBase extends Laya.Box {
    public btnBack!: Laya.Button;
    public btnScroll!: Laya.Button;
    public progress!: Laya.ProgressBar;
    public imgHead!: Laya.Image;
    public imgAward!: Laya.Image;
    public boxMask!: Laya.Box;
    public btnContinue!: Laya.Button;
    public lblDistance!: Laya.Label;
}
const { regClass } = Laya;
import { Game } from "../Game";
import { LevelEvent } from "../level/LevelConst";
import { LevelModel } from "../level/LevelModel";
import { SkinModel } from "../skin/SkinModel";
import { HudViewRTBase } from "./HudViewRT.generated";
@regClass()
export class HudViewRT extends HudViewRTBase {
    private updateView(): void {
        this.updateDistance();
        this.updateProgress();
        this.updateScrollButton();
    }
    private updateProgress(): void {
        const isShowProgress = LevelModel.ins.isShowProgress();
        this.progress.visible = isShowProgress;
        if (isShowProgress) {
            this.imgHead.skin = SkinModel.ins.getCurrentSkinHead();
            this.progress.value = LevelModel.ins.currDistanceFormat * 0.01;
            this.imgHead.x = this.progress.width * this.progress.value;
        }
    }
    private updateDistance(): void {
        const isPractice = LevelModel.ins.isPracticeMode();
        this.lblDistance.visible = isPractice;
        if (isPractice) {
            const distance = LevelModel.ins.currDistanceFormat;
            this.lblDistance.text = `当前距离：${distance}`;
        }
    }
    private updateScrollButton(): void {
        this.btnScroll.visible = LevelModel.ins.isExistScrollButton();
    }
    private onClickBack(): void {
        Game.ins.pause();
    }
    private onClickScroll(): void {
        Game.ins.scrollTo(LevelModel.ins.currTopDistance);
    }
    private onClickContinue(): void {
        this.btnBack.visible = true;
        this.boxMask.visible = false;
    }
    onAwake(): void {
        this.btnBack.on(Laya.Event.CLICK, this, this.onClickBack);
        this.btnScroll.on(Laya.Event.CLICK, this, this.onClickScroll);
        this.btnContinue.on(Laya.Event.CLICK, this, this.onClickContinue);
    }
    onEnable(): void {
        if (LevelModel.ins.isScrollClose) {
            this.btnBack.visible = false;
            this.boxMask.visible = true;
        }
        this.updateView();
        LevelModel.ins.on(LevelEvent.DistanceChanged, this, this.updateView);
    }
    onDisable(): void {
        LevelModel.ins.off(LevelEvent.DistanceChanged, this, this.updateView);
    }
}
import { ConfigPath } from "../../const/ConfigPath";
export interface ILevelPrefabData {
    name: string,
    visible: boolean,
    x: number,
    y: number,
    width: number,
    height: number,
    anchorX: number,
    anchorY: number,
    _$prefab: string
}
export class LevelEvent {
    static readonly DistanceChanged = "DistanceChanged";
    static readonly TopDistanceChanged = "TopDistanceChanged";
}
export enum ELevelMode {
    Practice,
    Normal,
}
export enum ELevelConst {
    LevelTestId = 0,
    LevelPracticeId = 10000,
    Level_10001 = 10001,
    Level_10002 = 10002,
}
export const LevelConfig = {
    [ELevelConst.LevelTestId]: { path: ConfigPath.LH_Level_Test },
    [ELevelConst.LevelPracticeId]: { path: ConfigPath.LH_Level_10000 },
    [ELevelConst.Level_10001]: { path: ConfigPath.LH_Level_10001 },
    [ELevelConst.Level_10002]: { path: ConfigPath.LH_Level_10002 },
}
export enum ELevelNodeSign {
    Item = "itemRoot",
    Ground = "groundRoot",
    Obstacle = "obstacleRoot",
}
export type ILevelParseProp<T extends Laya.Component> = {
    name: string;
    root: Laya.Sprite;
    components: T[],
    component: new () => T;
}
/**
 * author: 陈秀齐
 * time: 2023/12/19 15:31:20
 * desc:
 */
import { LocalData } from "../../utils/LocalData";
export type ILevelLocalData = { [key: number]: { topScore: number } }
export class LevelLocalData extends LocalData<ILevelLocalData> {
    static readonly Key = "LevelLocalData";
    static readonly Default: ILevelLocalData = {};
    constructor() {
        super(LevelLocalData.Key, LevelLocalData.Default);
    }
    newHistoryRecord(levelId: number, score: number): void {
        if (!this.data[levelId]) {
            this.data[levelId] = { topScore: score };
        } else {
            this.data[levelId].topScore = score;
        }
        this.save();
    }
    getHistoryRecord(levelId: number): number {
        return this.data[levelId] ? this.data[levelId].topScore : 0;
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/19 15:28:18
 * desc:
 */
import { Model } from "../../core/mvc/Model";
import { ConfigUtils, DialogConfigData } from "../../utils/ConfigUtils";
import { ELevelConst, LevelEvent } from "./LevelConst";
import { LevelLocalData } from "./LevelLocalData";
export class LevelModel extends Model {
    private static _ins: LevelModel;
    static get ins(): LevelModel {
        if (!this._ins) {
            this._ins = new LevelModel();
        }
        return this._ins;
```

```
}
private _distanceRatio = 100 / 16000;
private _startSpace = 0;
private _currId: number;
private _freeJumpTimes: number;
private _isScrollClose: boolean;
private _localData: LevelLocalData;
/** 当前关卡当前距离 */
private _currDistance: number = 0;
/** 当前关卡最高距离 */
private _currTopDistance: number = 0;
private _dialogIndex = 0;
private _enterLevelCount = 0;
private constructor() {
    super();
    this._localData = new LevelLocalData();
}
get currId(): number {
    return this._currId;
}
set currId(v: number) {
    this._currId = v;
    this._dialogIndex = 0;
    this._freeJumpTimes = 1;
    this._enterLevelCount++;
}
get skin(): number {
    return this._enterLevelCount <= 1 ? 0 : Math.floor(Math.random() * 3);
}
get isScrollClose(): boolean {
    return this._isScrollClose;
}
set isScrollClose(v: boolean) {
    this._isScrollClose = v;
}
set currDistance(v: number) {
    this._currDistance = v;
    this.event(LevelEvent.DistanceChanged, v);
    if (this._currDistance > this.currTopDistance) {
        this.currTopDistance = v;
        this.event(LevelEvent.TopDistanceChanged, v);
    }
}
get currDistance(): number {
    return this._currDistance;
}
set currTopDistance(v: number) {
    this._currTopDistance = v;
}
get currTopDistance(): number {
        return this._currTopDistance;
    }
    get currHistoryTopDistance(): number {
        return this._localData.getHistoryRecord(this._currId);
    }
    get currDistanceFormat(): number {
        return this.formatDistance(this.currDistance);
    }
    get currTopDistanceFormat(): number {
        return this.formatDistance(this.currTopDistance);
    }
    get currHistoryTopDistanceFormat(): number {
        return this.formatDistance(this.currHistoryTopDistance);
    }
    private formatDistance(distance: number): number {
        const real = Math.floor(distance * this._distanceRatio);
        return Math.max(real, 0);
    }
    setStartSpace(v: number): void {
        this._startSpace = v;
    }
    isPracticeMode(): boolean {
        return this.currId == ELevelConst.LevelPracticeId;
    }
    isSecondLevel(): boolean {
        return this.currId == ELevelConst.Level_10002;
    }
    isShowProgress(): boolean {
        return this.isSecondLevel() && this.currDistance > this._startSpace;
    }
    isExistTop(): boolean {
        return this.isSecondLevel() && this.currTopDistance > this.currDistance;
    }
    isExistScrollButton(): boolean {
        return this.isExistFree() && (this.isPracticeMode() || this.isSecondLevel()) &&
this.currTopDistance > this.currDistance;
    }
    isExistFree(): boolean {
        return this._freeJumpTimes > 0;
    }
    scrollEnd(): void {
        this._freeJumpTimes--;
        this.recordPlayerPos(this.currTopDistance);
    }
    recordPlayerPos(distance: number): void {
        this.currDistance = distance;
    }
    resetDistance(): void {
        this.currDistance = this.currTopDistance = 0;
    }
    checkExistNewRecord(): boolean {
        return this.currTopDistanceFormat > this.currHistoryTopDistanceFormat;
    }
    saveNewRecord(): void {
        this._localData.newHistoryRecord(this._currId, this.currTopDistance);
    }
    setLableDialog(label: Laya.Label): void {
        const configs: DialogConfigData[] = ConfigUtils.get("dialog");
        const config = configs.find(c => c.id == this._currId);
        if (!config) return;
        let dialog = config.dialogs[this._dialogIndex];
        this._dialogIndex = (this._dialogIndex + 1) % config.dialogs.length;
        label.text = dialog.desc;
        switch (dialog.type) {
            case 1:
                label.setVar("n", this.currHistoryTopDistanceFormat);
                break;
            case 2:
                label.setVar("n", this.currHistoryTopDistanceFormat - this.currDistanceFormat);
                break;
            default:
                break;
        }
    }
}
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/LoadingView.lh
 */
export class LoadingViewRTBase extends Laya.Box {
    public progress!: Laya.ProgressBar;
    public lblProgress!: Laya.Label;
}
const { regClass } = Laya;
import { StringUtils } from "../../utils/StringUtils";
import { LoadingViewRTBase } from "./LoadingViewRT.generated";
@regClass()
export class LoadingViewRT extends LoadingViewRTBase {
    private _desc: string = "100000";
    private _value: number;
    public get value(): number {
        return this._value;
    }
    public set value(v: number) {
        this._value = v;
        this.progress.value = v;
        this.lblProgress.text = `${StringUtils.lang(this.desc)}...${StringUtils.toPercent(v)}`;
    }
    public get desc(): string {
        return this._desc;
    }
    public set desc(v: string) {
        this._desc = v;
    }
}
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/MainView.lh
 */
export class MainViewRTBase extends Laya.Box {
    public btnPlay!: Laya.Button;
    public btnSkin!: Laya.Button;
    public btnTest!: Laya.Button;
    public btnHelp!: Laya.Button;
}
const { regClass } = Laya;
import { ConfigPath } from "../../const/ConfigPath";
import { ViewMgr } from "../../core/UI/ViewMgr";
import { Game } from "../Game";
import { EViewKey } from "../ViewConst";
import { ELevelConst } from "../level/LevelConst";
import { MainViewRTBase } from "./MainViewRT.generated";
@regClass()
export class MainViewRT extends MainViewRTBase {
    private onClickHelp(): void {
        ViewMgr.ins.open(EViewKey.HelpView);
    }
    private onClickSkin(): void {
        ViewMgr.ins.open(EViewKey.SkinView);
    }
    private onClickPlay(): void {
        Game.ins.enterLevel(ELevelConst.Level_10001);
    }
    private onClickTest(): void {
        Game.ins.enterLevel(ELevelConst.LevelTestId);
    }
    onAwake(): void {
        this.btnHelp.on(Laya.Event.CLICK, this.onClickHelp);
        this.btnSkin.on(Laya.Event.CLICK, this.onClickSkin);
        this.btnPlay.on(Laya.Event.CLICK, this.onClickPlay);
        this.btnTest.on(Laya.Event.CLICK, this.onClickTest);
    }
}
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/PauseView.lh
 */
export class PauseViewRTBase extends Laya.Box {
    public lblTitle!: Laya.Label;
    public btnClose!: Laya.Button;
    public boxPractice!: Laya.Box;
    public imgNewRecord!: Laya.Image;
    public lblCurrentScore!: Laya.Label;
    public lblHistoryScore!: Laya.Label;
    public boxNormal!: Laya.Box;
    public lblLeftScore!: Laya.Label;
    public imgIcon!: Laya.Image;
    public btnResume!: Laya.Button;
    public btnMainMenu!: Laya.Button;
    public btnRestart!: Laya.Button;
}
const { regClass } = Laya;
import { Game } from "../Game";
import { LevelModel } from "../level/LevelModel";
import { SkinModel } from "../skin/SkinModel";
import { PauseViewRTBase } from "./PauseViewRT.generated";
@regClass()
export class PauseViewRT extends PauseViewRTBase {
    private updateView(): void {
        let isPracticeMode = LevelModel.ins.isPracticeMode();
        this.boxNormal.visible = !isPracticeMode;
        this.boxPractice.visible = isPracticeMode;
        if (isPracticeMode) {
            let isNewRecord = LevelModel.ins.checkExistNewRecord();
            this.imgNewRecord.visible = isNewRecord;
            this.lblCurrentScore.text = LevelModel.ins.currTopDistanceFormat.toString();
            isNewRecord && LevelModel.ins.saveNewRecord();
            this.lblHistoryScore.text = LevelModel.ins.currHistoryTopDistanceFormat.toString();
        } else {
            this.imgIcon.skin = SkinModel.ins.getCurrentSkin();
        }
    }
    private onClickResume(): void {
        Game.ins.resume();
    }
    private onClickRestart(): void {
        Game.ins.restartLevel();
    }
    private onClickMainMenu(): void {
        Game.ins.backHome();
    }
    onAwake(): void {
        this.btnClose.on(Laya.Event.CLICK, this.onClickResume);
        this.btnResume.on(Laya.Event.CLICK, this.onClickResume);
        this.btnRestart.on(Laya.Event.CLICK, this.onClickRestart);
        this.btnMainMenu.on(Laya.Event.CLICK, this.onClickMainMenu);
    }
    onEnable(): void {
        this.updateView();
    }
}
export enum ESkinItemStatus {
    Locked,
    Idle,
    Adventure,
}
export class SkinEvent {
    static readonly Unlcok = "Unlcok";
    static readonly Adventure = "Adventure";
}
export interface ISkinListData {
    id: string;
    lblName: string;
    imgAvatar: string;
    status: ESkinItemStatus;
}
import { Controller } from "../../core/mvc/Controller";
import { SkinModel } from "./SkinModel";
/**
 * author: 陈秀齐
 * time: 2023/12/09 19:32:14
 * desc:
 */
export class SkinController extends Controller {
    private static _ins: SkinController;
    public static get ins(): SkinController {
        if (this._ins == null) {
            this._ins = new SkinController();
        }
        return this._ins;
    }
    private model: SkinModel;
    private constructor() {
        super();
        this.model = SkinModel.ins;
    }
    unlcok(id: string): void {
        this.model.unlock(id);
    }
    adventure(id: string): void {
        this.model.adventure(id);
    }
}
const { regClass } = Laya;
import { PathUtils } from "../../utils/PathUtils";
import { ESkinItemStatus } from "./SkinConst";
import { SkinItemRTBase } from "./SkinItemRT.generated";
@regClass()
export class SkinItemRT extends SkinItemRTBase {
    get dataSource(): any {
        return super.dataSource;
    }
    set dataSource(value: any) {
        super.dataSource = value;
        if (!value) return;
        let status = value.status;
        if (status != null) {
            let isIdle = status == ESkinItemStatus.Idle;
            let isLocked = status == ESkinItemStatus.Locked;
            let isWorking = status == ESkinItemStatus.Adventure;
            this.imgBg.skin = PathUtils.getUiImage(isLocked ? "com_box_2" : "com_box_1");
            this.imgAvatarBg.skin = PathUtils.getUiImage(isLocked ? "img_avatar_bg_2" : "img_avatar_bg_1");
            this.imgVideo.visible = isLocked;
            this.imgAdventuring.visible = isWorking;
            this.btnUnlock.visible = isLocked;
            this.btnAdventure.visible = isIdle;
        }
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/12 10:16:01
 * desc:
 */
import { LocalData } from "../../utils/LocalData";
export interface ISkinLocalData {
    skinId: string;
    idleSkinIds: string[];
}
export class SkinLocalData extends LocalData<ISkinLocalData> {
    static readonly Key = "SkinLocalData";
    static readonly Default: ISkinLocalData = {
        skinId: "100001",
        idleSkinIds: ["100001"],
    }
    constructor() {
        super(SkinLocalData.Key, SkinLocalData.Default);
    }
    unlock(id: string): void {
        this.data.idleSkinIds.push(id);
        this.save();
    }
    adventure(id: string): void {
        this.data.skinId = id;
        this.save();
    }
}
import { Model } from "../../core/mvc/Model";
import { ConfigUtils, SkinConfigData } from "../../utils/ConfigUtils";
import { PathUtils } from "../../utils/PathUtils";
import { ESkinItemStatus, ISkinListData, SkinEvent } from "./SkinConst";
import { SkinLocalData } from "./SkinLocalData";
/**
 * author: 陈秀齐
 * time: 2023/12/09 19:31:50
 * desc:
 */
export class SkinModel extends Model {
    private static _ins: SkinModel;
    public static get ins(): SkinModel {
        if (!this._ins) {
            this._ins = new SkinModel();
        }
        return this._ins;
    }
    private _localData: SkinLocalData;
    private constructor() {
        super();
        this._localData = new SkinLocalData();
    }
    checkStatus(id: string): ESkinItemStatus {
        let localData = this._localData.data;
        return localData.skinId === id ? ESkinItemStatus.Adventure : localData.idleSkinIds.some(idleId => idleId === id) ? ESkinItemStatus.Idle : ESkinItemStatus.Locked
    }
    getList(): ISkinListData[] {
        let arr: ISkinListData[] = [];
        const configs: SkinConfigData[] = ConfigUtils.get("skin");
        for (let i = 0; i < configs.length; i++) {
            const conf = configs[i];
            arr.push({
                id: conf.id,
                lblName: conf.name,
                status: this.checkStatus(conf.id),
                imgAvatar: PathUtils.getAvatar(conf.icon),
            });
        }
        return arr;
    }
    unlock(id: string): void {
        const configs: SkinConfigData[] = ConfigUtils.get("skin");
        if (!configs.some(conf => conf.id === id)) {
            console.warn("unlock fail!!!, this skin is not exist!", id);
            return;
        }
        this._localData.data.idleSkinIds.push(id);
        this._localData.unlock(id);
        this.event(SkinEvent.Unlcok, id);
    }
    adventure(id: string): void {
        const configs: SkinConfigData[] = ConfigUtils.get("skin");
        if (!configs.some(conf => conf.id === id)) {
            console.warn("unlock fail!!!, this skin is not exist!", id);
            return;
        }
        this._localData.adventure(id);
        this.event(SkinEvent.Adventure, id);
    }
    getCurrentSkin(): string {
        let id = this._localData.data.skinId;
        const configs: SkinConfigData[] = ConfigUtils.get("skin");
        let conf = configs.find(conf => conf.id === id);
        return PathUtils.getAvatar(conf.icon);
    }
    getCurrentSkinHead(): string {
        let id = this._localData.data.skinId;
        const configs: SkinConfigData[] = ConfigUtils.get("skin");
        let conf = configs.find(conf => conf.id === id);
        return PathUtils.getHead(conf.icon);
    }
}
/**This class is automatically generated by LayaAirIDE, please do not make any modifications. */
/**
 * resources/prefabs/views/SkinView.lh
 */
export class SkinViewRTBase extends Laya.Box {
    public lblTitle!: Laya.Label;
    public list!: Laya.List;
    public btnClose!: Laya.Button;
}
const { regClass } = Laya;
import { ViewMgr } from "../../core/UI/ViewMgr";
import { EViewKey } from "../ViewConst";
import { ESkinItemStatus, ISkinListData, SkinEvent } from "./SkinConst";
import { SkinController } from "./SkinController";
import { SkinModel } from "./SkinModel";
import { SkinViewRTBase } from "./SkinViewRT.generated";
@regClass()
export class SkinViewRT extends SkinViewRTBase {
    private onClickClose(): void {
        ViewMgr.ins.close(EViewKey.SkinView);
    }
    private onItemMouse(e: Laya.Event, index: number): void {
        if (e.type == Laya.Event.CLICK) {
            let listArray = this.list.array as ISkinListData[];
            let itemData = listArray[index];
            if (e.target.name == "btnAdventure") {
                // ctrl 选择皮肤
                SkinController.ins.adventure(itemData.id);
            } else if (e.target.name == "btnUnlock") {
                // ctrl 播放广告解锁皮肤
                SkinController.ins.unlcok(itemData.id);
            }
        }
    }
    private refreshStatus(id: string): void {
        let listArray = this.list.array as ISkinListData[];
        let index = listArray.findIndex(data => data.id === id);
        let itemData = listArray[index];
        itemData.status = SkinModel.ins.checkStatus(id);
        this.list.changeItem(index, itemData);
    }
    private onSkinAdventure(id: string): void {
        let listArray = this.list.array as ISkinListData[];
        let currAdventure = listArray.find(data => data.status === ESkinItemStatus.Adventure);
        this.refreshStatus(currAdventure.id);
        this.refreshStatus(id);
    }
    private onSkinUnlock(id: string): void {
        this.refreshStatus(id);
    }
    onAwake(): void {
        this.btnClose.on(Laya.Event.CLICK, this.onClickClose);
        this.list.mouseHandler = new Laya.Handler(this, this.onItemMouse, null, false);
    }
    onEnable(): void {
        SkinModel.ins.on(SkinEvent.Unlcok, this, this.onSkinUnlock);
        SkinModel.ins.on(SkinEvent.Adventure, this, this.onSkinAdventure);
        this.list.array = SkinModel.ins.getList();
    }
    onDisable(): void {
        SkinModel.ins.off(SkinEvent.Unlcok, this, this.onSkinUnlock);
        SkinModel.ins.off(SkinEvent.Adventure, this, this.onSkinAdventure);
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/29 10:11:17
 * desc:
 */
export class ArrayUtils {
    static intersection<T>(arrA: T[], arrB: T[]): T[] {
        return arrA.filter(i => arrB.indexOf(i) !== -1);
    }
    static difference<T>(arrA: T[], arrB: T[]): T[] {
        return arrA.filter(i => arrB.indexOf(i) === -1);
    }
    static union<T>(arr1: T[], arr2: T[]): T[] {
        return [...new Set([...arr1, ...arr2])];
    }
    static clear(arr: any[]): void {
        arr.length = 0;
    }
}
/**
```

```
 * author: 陈秀齐
 * time: 2023/12/11 08:57:49
 * desc:
 * TODO:
 * 1.参数 key 没有被约束；
 * 2.目前是通过 json 进行加载，需要支持压缩包以及二进制；
 * 3.解析出来目前是 any 类型的数据；
 */
import { ConfigPath } from "../const/ConfigPath";
import { TConstructor } from "../core/base/CoreConst";
export class SkinConfigData {
    id: string;
    name: string;
    icon: string;
    constructor(data: any) {
        this.id = data.id;
        this.name = data.name;
        this.icon = data.icon;
    }
}
export class DialogConfigData {
    id: number;
    dialogs: { type: number, desc: string }[];
    constructor(data: any) {
        this.id = data.id;
        this.dialogs = data.dialogs;
    }
}
export interface IConfigData {
    path: string,
    cls: TConstructor
}
export const ConfigUtilsMap: { [key: string]: IConfigData } = {
    skin: {
        path: ConfigPath.JSON_Skin,
        cls: SkinConfigData
    },
    dialog: {
        path: ConfigPath.JSON_Dialog,
        cls: DialogConfigData
    },
}
export class ConfigUtils {
    private static jsonMap: { [key: string]: any } = {};
    static get(key: keyof typeof ConfigUtilsMap) {
        if (this.jsonMap[key] == null) {
            const { cls, path } = ConfigUtilsMap[key];
            let res = Laya.loader.getRes(path);
            if (res) {
                let arr = [];
                for (let key in res.data) {
                    let data = new cls(res.data[key]);
                    arr.push(data);
                }
                this.jsonMap[key] = arr;
            }
        }
        return this.jsonMap[key];
    }
}
import { LocalStorageUtils } from "./LocalStorageUtils";
export class LocalData<DATATYPE extends object> {
    private _key: string;
    private _data: DATATYPE;
    private _defaultData: DATATYPE;
    constructor(key: string, defaultData: DATATYPE) {
        this._key = key;
        this._defaultData = defaultData;
    }
    get data(): DATATYPE {
        if (this._data == null) {
            this._data = LocalStorageUtils.load(this._key);
            if (this._data == null) {
                this._data = this._defaultData;
            }
        }
        return this._data;
    }
    save(): void {
        LocalStorageUtils.save(this._key, this.data);
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/12 09:31:01
 * desc: 本地数据储存
 * TODO:
 * 1.数据校验，防止篡改和失效；如字段缺失，或违法
 */
export class LocalStorageUtils {
    static _game: any;
    private static GAME_KEY: string = "_game_";
    static get game(): any {
        if (this._game == null) {
            this._game = Laya.LocalStorage.getJSON(LocalStorageUtils.GAME_KEY) || {};
        }
        return this._game;
    }
    private static saveGame(): void {
        console.log("saveGame=========", this.game);
        Laya.LocalStorage.setJSON(LocalStorageUtils.GAME_KEY, this.game);
    }
    static save(key: string, data: any): void {
        this.game[key] = data;
        Laya.CallLater.I.callLater(this, this.saveGame);
    }
    static saveNow(key: string, data: any): void {
        this.game[key] = data;
        Laya.CallLater.I.runCallLater(this, this.saveGame);
    }
    static load(key: string): any {
        return this.game[key];
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/09 19:14:46
 * desc:
 */
export class MathUtil {
    /**
     * int 的最大值
     */
    public static INT_MAX_VALUE: number = 2147483647;
    /**
     * 一弧度的角度数
     */
    public static ONE_RADIANS: number = 180 / Math.PI;
    /**
     * 弧度转换成角度
     * @param radians
     * @return
     */
    public static radiansToDegrees(radians: number): number {
        return radians * this.ONE_RADIANS;
    }
    /**
     * 角度转换成弧度
     * @param degrees
     * @return
     */
    public static degreesToRadians(degrees: number): number {
        return (degrees * Math.PI) / 180;
    }
    /**
     * 得到一个区间的随机数
     * @param min 最小数
     * @param max 最大数
     */
    public static randomF(min: number, max: number): number {
        return min + Math.random() * (max - min);
    }
    /**
     * 得到一个区间的随机整数,结果包含最小数跟最大数
     * @param min 最小数
     * @param max 最大数
     */
    public static randomI(min: number, max: number): number {
        return this.randomF(min, max + 0.99999) >> 0;
    }
    /**
     * 掷硬币 50%
     * @returns
     */
    public static coinFlip(): boolean {
        return this.randomF(0, 1) > 0.5;
    }
    /**
     * 得到一个数组的随机项
     * @param list
     * @returns
     */
    public static randElement<T>(list: T[]): T | null {
        if (list == null || list.length == 0) {
            return null;
        }
        return list[this.randomI(0, list.length - 1)];
    }
    /**
     * 概率是否发生
     * @param value (0~1)
     * @returns
     */
    public static chance(value: number): boolean {
        return Math.random() < value;
    }
    /**
     * 判断两区间是否部分重叠
     * @param rangeAMin 区间 0 起始值
     * @param rangeAMax 区间 0 结束值
     * @param rangeBMin 区间 1 起始值
     * @param rangeBMax 区间 1 起始值
     * @returns
     */
    public static isPartiallyOverlap(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {
        return rangeAMin <= rangeBMax && rangeAMax >= rangeBMin;
    }
    // 判断一个小数区间是否包括另一个小数区间
    public static isIntervalIncluding(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {
        return rangeAMin <= rangeBMin && rangeAMax >= rangeBMax;
    }
    // 判断两个小数区间是否不相交也不包括
    public static isIntervalsDisjoint(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {
        return rangeAMax < rangeBMin || rangeAMin > rangeBMax;
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/12 09:40:48
 * desc:
 */
export class ObjectUtils {
    static isEmpty(obj: any): boolean {
        return Object.keys(obj).length === 0;
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/20 15:56:33
 * desc:
 */
export class PathUtils {
    static getAvatar(icon: string): string {
        return `resources/icon/avatar/${icon}.png`;
    }
    static getHead(icon: string): string {
        return `resources/icon/avatar/${icon}_head.png`;
    }
    static getUiImage(icon: string): string {
        return `atlas/ui/${icon}.png`;
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/07 19:45:28
 * desc:
 */
export class StringUtils {
    /**
     * 转换成百分比
     * @param decimal 小数
     * @param precision 精度位数 默认两位
     */
    static toPercent(decimal: number, precision: number = 2): string {
        return `${(decimal * 100).toFixed(precision)}%`;
    }
    static lang(key: string): string {
        return Laya.Text.langPacks ? Laya.Text.langPacks[key] : key;
    }
    static format(template: string, ...values: string[]): string {
        return template.replace(/{(\d+)}/g, (match, index) => {
            let replacement = values[Number(index)];
            replacement = this.lang(replacement);
            return typeof replacement !== 'undefined' ? replacement : match;
        });
    }
}
export class SegmentTree {
    root: SegmentTreeNode | null;
    constructor() {
        this.root = null;
    }
    // 构建线段树
    buildTree(ranges: IRange[]): void {
        if (ranges.length === 0) {
            return;
        }
        const min = Math.min(...ranges.map(r => r.min));
        const max = Math.max(...ranges.map(r => r.max));
        this.root = this._buildTree(ranges, min, max);
    }
    private _buildTree(ranges: IRange[], min: number, max: number): SegmentTreeNode | null {
        if (min > max) {
            return null;
        }
        const root = new SegmentTreeNode(min, max);
        if (min < max) {
            const mid = Math.floor((min + max) / 2);
            root.left = this._buildTree(ranges, min, mid);
            root.right = this._buildTree(ranges, mid + 1, max);
        }
        // 找到与当前节点区间相交的区间
        root.intersects = ranges.filter(
            r => r.min <= root.max && r.max >= root.min
        );
        return root;
    }
    // 查询与给定区间相交的区间
    query(root: SegmentTreeNode | null, min: number, max: number): IRange[] {
        const result: IRange[] = [];
        if (root === null || min > root.max || max < root.min) {
            return result;
        }
        if (min <= root.min && max >= root.max) {
            // 如果当前节点区间完全包含在查询区间内，则返回当前节点的相交区间
            return root.intersects;
        }
        // 否则，递归查询左右子树
        result.push(...this.query(root.left, min, max));
        result.push(...this.query(root.right, min, max));
        return [...new Set(result)];
    }
    clear(): void {
        this.root = null;
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/29 19:09:06
 * desc:
 */
export interface IRange {
    min: number;
    max: number;
}
export class SegmentTreeNode {
    min: number;
    max: number;
    left: SegmentTreeNode | null;
    right: SegmentTreeNode | null;
    intersects: IRange[];
    constructor(min: number, max: number) {
        this.min = min;
        this.max = max;
        this.left = null;
        this.right = null;
        this.intersects = [];
    }
}
import { Model } from "./Model";
import { Controller } from "./Controller";
import { TConstructor } from "../base/CoreConst";
export type MOC = Model | Controller;
export class MVCDecorator {
    static _classMap_: Map<string, TConstructor<MOC>> = new Map();
    static _instanceMap_: Map<string, MOC> = new Map();
    static reg(cls: any): void {
        const clsName = (cls as any).name;
        if (!(cls.prototype instanceof Controller) && !(cls.prototype instanceof Model)) {
            console.warn("mvc reg warning!!! register a class not model or ctrl", clsName, cls);
            return;
        }
        if (MVCDecorator._classMap_.has(clsName)) {
            console.warn("mvc reg warning!!! repeat register", clsName, cls);
            return;
        }
        MVCDecorator._classMap_.set(clsName, cls);
    }
    static prop(cls: Controller | Model): PropertyDecorator {
        return (target: any, key: PropertyKey) => {
            const getter = function () {
                const clsName = (cls as any).name;
                if (!MVCDecorator._classMap_.has(clsName)) {
                    console.warn("mvc prop warning!!! try to get a model no register", clsName, key, cls);
                    return;
                }
                const cacheClass = MVCDecorator._classMap_.get(clsName);
                let instance = MVCDecorator._instanceMap_.get(clsName);
                if (!MVCDecorator._instanceMap_.has(clsName)) {
                    instance = new cacheClass();
                    MVCDecorator._instanceMap_.set(clsName, instance);
                }
                return instance;
            };

            const setter = function () {
                const clsName = (cls as any).name;
                console.warn("mvc prop warning!!! try to set a prop.", clsName, key, cls);
            };
            // Redefine the property with the new getter and setter
            Object.defineProperty(target, key, {
              get: getter,
              set: setter,
              enumerable: true,
              configurable: true,
            });
        }
    }
    // todo
    private static model_handler(cls: Model, event: string) {
        return (target: Object, propertyKey: string | symbol, descriptor: any) => {
            console.log("model_handler========", target, propertyKey, descriptor);
        }
    }
}
import { NodeFlags } from "../Const"
import { Component } from "../components/Component"
import { Event } from "../events/Event"
import { EventDispatcher } from "../events/EventDispatcher"
import { Pool } from "../utils/Pool"
import { Stat } from "../utils/Stat"
import { Timer } from "../utils/Timer"
import { ILaya } from "../../ILaya";
import { ComponentDriver } from "../components/ComponentDriver";
const ARRAY_EMPTY: any[] = [];
```

```
export type Callback = ((...args: any[]) => Promise<void>) | ((...args: any[]) => void) | undefined;
export interface ITransition<STATE, EVENT, CALLBACK> {
    fromState: STATE;
    event: EVENT;
    toState: STATE;
    cb: CALLBACK;
}
export function t<STATE, EVENT, CALLBACK>(
    fromState: STATE, event: EVENT, toState: STATE,
    cb?: CALLBACK): ITransition<STATE, EVENT, CALLBACK> {
    return { fromState, event, toState, cb };
}
/**
 * <p> <code>Pool</code> 是对象池类，用于对象的存储、重复使用。</p>
 * <p>合理使用对象池，可以有效减少对象创建的开销，避免频繁的垃圾回收，从而优化游戏流畅度。</p>
 */
export class Pool {
    /**@private */
    private static _CLSID: number = 0;
    /**@private */
    private static POOLSIGN: string = "__InPool";
    /**@private  对象存放池。 */
    private static _poolDic: any = {};
    /**
     * 根据对象类型标识字符，获取对象池。
     * @param sign 对象类型标识字符。
     * @return 对象池。
     */
    static getPoolBySign(sign: string): any[] {
        return Pool._poolDic[sign] || (Pool._poolDic[sign] = []);
    }
    /**
     * 清除对象池的对象。
     * @param sign 对象类型标识字符。
     */
    static clearBySign(sign: string): void {
        if (Pool._poolDic[sign]) Pool._poolDic[sign].length = 0;
    }
    /**
     * 将对象放到对应类型标识的对象池中。
     * @param sign 对象类型标识字符。
     * @param item 对象。
     */
    static recover(sign: string, item: any): void {
        if (item[Pool.POOLSIGN] !== false) //有这个标志，才表明对象是从 Pool 里获取的，允许 recover
            return;
        item[Pool.POOLSIGN] = true;
        Pool.getPoolBySign(sign).push(item);
    }
    /**
     * 根据类名进行回收，如果类有类名才进行回收，没有则不回收
     * @param    instance 类的具体实例
     */
    static recoverByClass(instance: any): void {
        if (instance) {
            var className: string = instance["__className"] || instance.constructor._$gid;
            if (className) Pool.recover(className, instance);
        }
    }
    /**
     * 返回类的唯一标识
     */
    private static _getClassSign(cla: any): string {
        var className = cla["__className"] || cla["_$gid"];
        if (!className) {
            cla["_$gid"] = className = Pool._CLSID + "";
            Pool._CLSID++;
        }
        return className;
    }
    /**
     * 根据类型创建对象
     * @param    cls 类型
     */
    static createByClass<T>(cls: new () => T): T {
        return Pool.getItemByClass(Pool._getClassSign(cls), cls);
    }
    /**
     * <p>根据传入的对象类型标识字符，获取对象池中此类型标识的一个对象实例。</p>
     * <p>当对象池中无此类型标识的对象时，则根据传入的类型，创建一个新的对象返回。</p>
     * @param sign 对象类型标识字符。
     * @param cls 用于创建该类型对象的类。
     * @return 此类型标识的一个对象。
     */
    static getItemByClass<T>(sign: string, cls: new () => T): T {
        let rst: any;
        let pool = Pool.getPoolBySign(sign);
        if (pool.length)
            rst = pool.pop();
        else
            rst = new cls();
        rst[Pool.POOLSIGN] = false;
        return rst;
    }
    /**
     * <p>根据传入的对象类型标识字符，获取对象池中此类型标识的一个对象实例。</p>
     * <p>当对象池中无此类型标识的对象时，则使用传入的创建此类型对象的函数，新建一个对象返回。</p>
     * @param sign 对象类型标识字符。
     * @param createFun 用于创建该类型对象的方法。
     * @param caller this 对象
     * @return 此类型标识的一个对象。
     */
    static getItemByCreateFun(sign: string, createFun: Function, caller: any = null): any {
        var pool: any[] = Pool.getPoolBySign(sign);
        var rst: any = pool.length ? pool.pop() : createFun.call(caller);
        rst[Pool.POOLSIGN] = false;
        return rst;
    }
    /**
     * 根据传入的对象类型标识字符，获取对象池中已存储的此类型的一个对象，如果对象池中无此类型的对象，则返回 null 。
     * @param sign 对象类型标识字符。
     * @return 对象池中此类型的一个对象，如果对象池中无此类型的对象，则返回 null 。
     */
    static getItem(sign: string): any {
        var pool: any[] = Pool.getPoolBySign(sign);
        var rst: any = pool.length ? pool.pop() : null;
        if (rst) {
            rst[Pool.POOLSIGN] = false;
        }
        return rst;
    }
}
/**
 * 添加到父对象后调度。
 * @eventType Event.ADDED
 */
/*[Event(name = "added", type = "laya.events.Event")]*/
/**
 * 被父对象移除后调度。
 * @eventType Event.REMOVED
 */
/*[Event(name = "removed", type = "laya.events.Event")]*/
/**
 * 加入节点树时调度。
 * @eventType Event.DISPLAY
 */
/*[Event(name = "display", type = "laya.events.Event")]*/
```

```
/**
 * 从节点树移除时调度。
 * @eventType Event.UNDISPLAY
 */
/*[Event(name = "undisplay", type = "laya.events.Event")]*/
/**
 * <code>Node</code> 类是可放在显示列表中的所有对象的基类。该显示列表管理Laya 运行时中显示的所有对象。使用 Node 类排列显示列表中的显示对象。Node 对象可以有子显示对象。
 */
export class Node extends EventDispatcher {
    static EVENT_SET_ACTIVESCENE: string = "ActiveScene";
    static EVENT_SET_IN_ACTIVESCENE: string = "InActiveScene";
    /**@private */
    private _bits: number = 0;
    /**@private */
    private _hideFlags: number = 0;
    /**@internal 子对象集合，请不要直接修改此对象。*/
    _children: Node[] = ARRAY_EMPTY;
    /**@internal 父节点对象*/
    _parent: Node = null;
    /**@internal */
    _destroyed: boolean = false;
    /**@internal */
    _conchData: any;
    /**@internal */
    _componentDriver: ComponentDriver;
    /**@internal */
    _is3D: boolean;
    _url: string;
    _extra: INodeExtra;
    /**节点名称。*/
    name: string = "";
    /** 节点标签 */
    tag: string;
    /**
     * 如果节点从资源中创建，这里记录是他的 url
     */
    get url(): string {
        return this._url;
    }
    /**
     * 设置资源的 URL
     */
    set url(path: string) {
        this._url = path;
    }
    get hideFlags(): number {
        return this._hideFlags;
```

```
}
set hideFlags(value: number) {
    this._hideFlags = value;
}
/** 是否 3D 节点，即 Scene3D 和 Sprite3D 及其衍生类 */
get is3D(): boolean {
    return this._is3D;
}
/** 是否已经销毁。对象销毁后不能再使用。*/
get destroyed(): boolean {
    return this._destroyed;
}
constructor() {
    super();
    this._initialize();
}
//@internal
_initialize(): void {
    this._extra = {};
}
_setBit(type: number, value: boolean): void {
    if (type === NodeFlags.DISPLAY) {
        var preValue: boolean = this._getBit(type);
        if (preValue != value) this._updateDisplayedInstage();
    }
    if (value) this._bits |= type;
    else this._bits &= ~type;
}
_getBit(type: number): boolean {
    return (this._bits & type) != 0;
}
/**@internal */
_setUpNoticeChain(): void {
    if (this._getBit(NodeFlags.DISPLAY)) this._setBitUp(NodeFlags.DISPLAY);
}
/**@internal */
_setBitUp(type: number): void {
    var ele: Node = this;
    ele._setBit(type, true);
    ele = ele._parent;
    while (ele) {
        if (ele._getBit(type)) return;
        ele._setBit(type, true);
        ele = ele._parent;
    }
}
protected onStartListeningToType(type: string) {
    if (type === Event.DISPLAY || type === Event.UNDISPLAY) {
        if (!this._getBit(NodeFlags.DISPLAY)) this._setBitUp(NodeFlags.DISPLAY);
    }
```

```
}
bubbleEvent(type: string, data?: any) {
    let arr: Array<Node> = _bubbleChainPool.length > 0 ? _bubbleChainPool.pop() : [];
    arr.length = 0;
    let obj: Node = this;
    while (obj) {
        if (obj.activeInHierarchy)
            arr.push(obj);
        obj = obj.parent;
    }
    if (data instanceof Event) {
        data._stopped = false;
        for (let obj of arr) {
            data.setTo(type, obj, this);
            obj.event(type, data);
            if (data._stopped)
                break;
        }
    }
    else {
        for (let obj of arr)
            obj.event(type, data);
    }
    _bubbleChainPool.push(arr);
}
hasHideFlag(flag: number): boolean {
    return (this._hideFlags & flag) != 0;
}
/**
 * <p>销毁此对象。destroy 对象默认会把自己从父节点移除，并且清理自身引用关系，等待 js 自动垃圾回收机制回收。destroy 后不能再使用。</p>
 * <p>destroy 时会移除自身的事情监听，自身的 timer 监听，移除子对象及从父节点移除自己。</p>
 * @param destroyChild （可选）是否同时销毁子节点，若值为 true,则销毁子节点，否则不销毁子节点。
 */
destroy(destroyChild: boolean = true): void {
    this._destroyed = true;
    this.destroyAllComponent();
    this._parent && this._parent.removeChild(this);
    //销毁子节点
    if (this._children) {
        if (destroyChild) this.destroyChildren();
        else this.removeChildren();
    }
    this.onDestroy();
    this._children = null;
    //移除所有事件监听
    this.offAll();
```

```
}
/**
 * 销毁时执行
 * 此方法为虚方法，使用时重写覆盖即可
 */
onDestroy(): void {
    //trace("onDestroy node", this.name);
}
/**
 * 销毁所有子对象，不销毁自己本身。
 */
destroyChildren(): void {
    //销毁子节点
    if (this._children) {
        //为了保持销毁顺序，所以需要正序销毁
        for (let i = 0, n = this._children.length; i < n; i++) {
            this._children[0] && this._children[0].destroy(true);
        }
    }
}
/**
 * 添加子节点。
 * @param       node 节点对象
 * @return返回添加的节点
 */
addChild<T extends Node>(node: T): T {
    if (!node || this._destroyed || node as any === this) return node;
    if ((<any>node)._zOrder) this._setBit(NodeFlags.HAS_ZORDER, true);
    if (node._parent === this) {
        var index: number = this.getChildIndex(node);
        if (index !== this._children.length - 1) {
            this._children.splice(index, 1);
            this._children.push(node);
            this._childChanged();
        }
    } else {
        node._parent && node._parent.removeChild(node);
        this._children === ARRAY_EMPTY && (this._children = []);
        this._children.push(node);
        node._setParent(this);
    }
    return node;
}
/**
 * 批量增加子节点
 * @param       ...args 无数子节点。
 */
addChildren(...args: any[]): void {
    var i: number = 0, n: number = args.length;
```

```
        while (i < n) {
            this.addChild(args[i++]);
        }
    }
    /**
     * 添加子节点到指定的索引位置。
     * @param       node 节点对象。
     * @param       index 索引位置。
     * @return返回添加的节点。
     */
    addChildAt(node: Node, index: number): Node {
        if (!node || this._destroyed || node === this) return node;
        if ((<any>node)._zOrder) this._setBit(NodeFlags.HAS_ZORDER, true);
        if (index >= 0 && index <= this._children.length) {
            if (node._parent === this) {
                var oldIndex: number = this.getChildIndex(node);
                this._children.splice(oldIndex, 1);
                this._children.splice(index, 0, node);
                this._childChanged();
            } else {
                node._parent && node._parent.removeChild(node);
                this._children === ARRAY_EMPTY && (this._children = []);
                this._children.splice(index, 0, node);
                node._setParent(this);
            }
            return node;
        } else {
            throw new Error("appendChildAt:The index is out of bounds");
        }
    }
    /**
     * 根据子节点对象，获取子节点的索引位置。
     * @param       node 子节点。
     * @return子节点所在的索引位置。
     */
    getChildIndex(node: Node): number {
        return this._children.indexOf(node);
    }
    /**
     * 根据子节点的名字，获取子节点对象。
     * @param       name 子节点的名字。
     * @return节点对象。
     */
    getChildByName(name: string): Node {
        for (let child of this._children) {
            if (child && child.name === name)
                return child;
        }
        return null;
```

```
}
/**
 * 根据子节点的索引位置，获取子节点对象。
 * @param      index 索引位置
 * @return子节点
 */
getChildAt(index: number): Node {
    return this._children[index] || null;
}
/**
 * 设置子节点的索引位置。
 * @param      node 子节点。
 * @param      index 新的索引。
 * @return返回子节点本身。
 */
setChildIndex(node: Node, index: number): Node {
    var childs: any[] = this._children;
    if (index < 0 || index >= childs.length) {
        throw new Error("setChildIndex:The index is out of bounds.");
    }
    var oldIndex: number = this.getChildIndex(node);
    if (oldIndex < 0) throw new Error("setChildIndex:node is must child of this object.");
    childs.splice(oldIndex, 1);
    childs.splice(index, 0, node);
    this._childChanged();
    return node;
}
/**
 * 子节点发生改变。
 * @private
 * @param      child 子节点。
 */
protected _childChanged(child: Node = null): void {
}
/**
 * 删除子节点。
 * @param      node 子节点
 * @return被删除的节点
 */
removeChild(node: Node): Node {
    if (!this._children) return node;
    var index: number = this._children.indexOf(node);
    return this.removeChildAt(index);
}
/**
 * 从父容器删除自己，如已经被删除不会抛出异常。
 * @return 当前节点（ Node ）对象。
 */
```

```
removeSelf(): Node {
    this._parent && this._parent.removeChild(this);
    return this;
}
/**
 * 根据子节点名字删除对应的子节点对象，如果找不到不会抛出异常。
 * @param      name 对象名字。
 * @return 查找到的节点 （ Node ） 对象。
 */
removeChildByName(name: string): Node {
    var node: Node = this.getChildByName(name);
    node && this.removeChild(node);
    return node;
}
/**
 * 根据子节点索引位置，删除对应的子节点对象。
 * @param      index 节点索引位置。
 * @return被删除的节点。
 */
removeChildAt(index: number): Node {
    var node: Node = this.getChildAt(index);
    if (node) {
        this._children.splice(index, 1);
        node._setParent(null);
    }
    return node;
}
/**
 * 删除指定索引区间的所有子对象。
 * @param      beginIndex 开始索引。
 * @param      endIndex 结束索引。
 * @return 当前节点对象。
 */
removeChildren(beginIndex: number = 0, endIndex: number = 0x7fffffff): Node {
    if (this._children && this._children.length > 0) {
        var childs: any[] = this._children;
        if (beginIndex === 0 && endIndex >= childs.length - 1) {
            var arr: any[] = childs;
            this._children = ARRAY_EMPTY;
        } else {
            arr = childs.splice(beginIndex, endIndex - beginIndex + 1);
        }
        for (var i: number = 0, n: number = arr.length; i < n; i++) {
            arr[i]._setParent(null);
        }
    }
    return this;
}
/**
```

```
 * 替换子节点。
 * 将传入的新节点对象替换到已有子节点索引位置处。
 * @param     newNode 新节点。
 * @param     oldNode 老节点。
 * @return返回新节点。
 */
replaceChild(newNode: Node, oldNode: Node): Node {
    var index: number = this._children.indexOf(oldNode);
    if (index > -1) {
        this._children.splice(index, 1, newNode);
        oldNode._setParent(null);
        newNode._setParent(this);
        return newNode;
    }
    return null;
}
/**
 * 子对象数量。
 */
get numChildren(): number {
    return this._children ? this._children.length : 0;
}
/**父节点。*/
get parent(): Node {
    return this._parent;
}
/**检查本节点是否是某个节点的上层节点
 * @param node
 * @return
 */
isAncestorOf(node: Node): boolean {
    let p = node.parent;
    while (p) {
        if (p == this)
            return true;
        p = p.parent;
    }
    return false;
};
/**@private */
protected _setParent(value: Node): void {
    if (this._parent !== value) {
        if (value) {
            this._parent = value;
            //如果父对象可见，则设置子对象可见
            this._onAdded();
            this.event(Event.ADDED);
            if (this._getBit(NodeFlags.DISPLAY)) {
                this._setUpNoticeChain();
                value.displayedInStage && this._displayChild(this, true);
            }
            value._childChanged(this);
        } else {
            //设置子对象不可见
            this._onRemoved();
            this.event(Event.REMOVED);
            let p = this._parent;
            if (this._getBit(NodeFlags.DISPLAY)) this._displayChild(this, false);
            this._parent = value;
            p._childChanged(this);
        }
    }
}
/**表示是否在显示列表中显示。*/
get displayedInStage(): boolean {
    if (this._getBit(NodeFlags.DISPLAY)) return this._getBit(NodeFlags.DISPLAYED_INSTAGE);
    this._setBitUp(NodeFlags.DISPLAY);
    return this._getBit(NodeFlags.DISPLAYED_INSTAGE);
}
/**@private */
private _updateDisplayedInstage(): void {
    var ele: Node;
    ele = this;
    var stage: Node = ILaya.stage;
    var displayedInStage: boolean = false;
    while (ele) {
        if (ele._getBit(NodeFlags.DISPLAY)) {
            displayedInStage = ele._getBit(NodeFlags.DISPLAYED_INSTAGE);
            break;
        }
        if (ele === stage || ele._getBit(NodeFlags.DISPLAYED_INSTAGE)) {
            displayedInStage = true;
            break;
        }
        ele = ele._parent;
    }
    this._setBit(NodeFlags.DISPLAYED_INSTAGE, displayedInStage);
}
/**@internal */
_setDisplay(value: boolean): void {
    if (this._getBit(NodeFlags.DISPLAYED_INSTAGE) !== value) {
        this._setBit(NodeFlags.DISPLAYED_INSTAGE, value);
        if (value) this.event(Event.DISPLAY);
        else this.event(Event.UNDISPLAY);
    }
}
/**
 * 设置指定节点对象是否可见(是否在渲染列表中)。
 * @private
```

```
     * @param      node 节点。
     * @param      display 是否可见。
     */
    private _displayChild(node: Node, display: boolean): void {
        var childs: any[] = node._children;
        if (childs) {
            for (var i: number = 0, n: number = childs.length; i < n; i++) {
                var child: Node = childs[i];
                if (!child) continue;
                if (!child._getBit(NodeFlags.DISPLAY)) continue;
                if (child._children.length > 0) {
                    this._displayChild(child, display);
                } else {
                    child._setDisplay(display);
                }
            }
        }
        node._setDisplay(display);
    }
    /**
     * 当前容器是否包含指定的 <code>Node</code> 节点对象 。
     * @param      node  指定的 <code>Node</code> 节点对象 。
     * @return 一个布尔值表示是否包含指定的 <code>Node</code> 节点对象 。
     */
    contains(node: Node): boolean {
        if (node === this) return true;
        while (node) {
            if (node._parent === this) return true;
            node = node._parent;
        }
        return false;
    }
    /**
     * 定时重复执行某函数。功能同 Laya.timer.timerLoop()。
     * @param      delay          间隔时间(单位毫秒)。
     * @param      caller         执行域(this)。
     * @param      method         结束时的回调方法。
     * @param      args        （可选）回调参数。
     * @param      coverBefore    （可选）是否覆盖之前的延迟执行，默认为 true。
     * @param      jumpFrame 时钟是否跳帧。基于时间的循环回调，单位时间间隔内，如能执行多次回调，出于性能考虑，引擎默认只执行一次，设置 jumpFrame=true 后，则回调会连续执行多次
     */
    timerLoop(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true, jumpFrame: boolean = false): void {
        this.timer.loop(delay, caller, method, args, coverBefore, jumpFrame);
    }
    /**
```

```
     * 定时执行某函数一次。功能同 Laya.timer.timerOnce()。
     * @param    delay        延迟时间(单位毫秒)。
     * @param    caller       执行域(this)。
     * @param    method       结束时的回调方法。
     * @param    args       （可选）回调参数。
     * @param    coverBefore  （可选）是否覆盖之前的延迟执行，默认为 true。
     */
    timerOnce(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {
        this.timer._create(false, false, delay, caller, method, args, coverBefore);
    }
    /**
     * 定时重复执行某函数(基于帧率)。功能同 Laya.timer.frameLoop()。
     * @param    delay        间隔几帧(单位为帧)。
     * @param    caller       执行域(this)。
     * @param    method       结束时的回调方法。
     * @param    args       （可选）回调参数。
     * @param    coverBefore  （可选）是否覆盖之前的延迟执行，默认为 true。
     */
    frameLoop(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {
        this.timer._create(true, true, delay, caller, method, args, coverBefore);
    }
    /**
     * 定时执行一次某函数(基于帧率)。功能同 Laya.timer.frameOnce()。
     * @param    delay        延迟几帧(单位为帧)。
     * @param    caller       执行域(this)
     * @param    method       结束时的回调方法
     * @param    args       （可选）回调参数
     * @param    coverBefore  （可选）是否覆盖之前的延迟执行，默认为 true
     */
    frameOnce(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {
        this.timer._create(true, false, delay, caller, method, args, coverBefore);
    }
    /**
     * 清理定时器。功能同 Laya.timer.clearTimer()。
     * @param    caller 执行域(this)。
     * @param    method 结束时的回调方法。
     */
    clearTimer(caller: any, method: Function): void {
        this.timer.clear(caller, method);
    }
    /**
     * <p>延迟运行指定的函数。</p>
     * <p>在控件被显示在屏幕之前调用，一般用于延迟计算数据。</p>
     * @param method 要执行的函数的名称。例如，functionName。
     * @param args 传递给 <code>method</code> 函数的可选参数列表。
     *
     * @see #runCallLater()
     */
    callLater(method: Function, args: any[] = null): void {
        this.timer.callLater(this, method, args);
    }
    /**
     * <p>如果有需要延迟调用的函数（通过 <code>callLater</code> 函数设置），则立即执行延迟调用函数。</p>
     * @param method 要执行的函数名称。例如，functionName。
     * @see #callLater()
     */
    runCallLater(method: Function): void {
        this.timer.runCallLater(this, method);
    }
    //==========================组件化支持============================
    /** @private */
    protected _components: Component[];
    /**@private */
    private _activeChangeScripts: Component[];
    _scene: Node;
    /**
     * 获得所属场景。
     * @return场景。
     */
    get scene(): any {
        return this._scene;
    }
    /**
     * 获取自身是否激活。
     *   @return    自身是否激活。
     */
    get active(): boolean {
        return !this._getBit(NodeFlags.NOT_READY) && !this._getBit(NodeFlags.NOT_ACTIVE);
    }
    /**
     * 设置是否激活。
     * @param    value 是否激活。
     */
    set active(value: boolean) {
        value = !!value;
        if (!this._getBit(NodeFlags.NOT_ACTIVE) !== value) {
            if (this._activeChangeScripts && this._activeChangeScripts.length !== 0) {
                if (value)
                    throw "Node: can't set the main inActive node active in hierarchy,if the operate is in main inActive node or it's children script's onDisable Event.";
                else
                throw "Node: can't set the main active node inActive in hierarchy,if the operate is in main active node or it's children script's onEnable Event.";
            } else {
                this._setBit(NodeFlags.NOT_ACTIVE, !value);
                if (this._parent) {
                    if (this._parent.activeInHierarchy) {
                        this._processActive(value, true);
                    }
                }
            }
        }
    }
    /**
     * 获取在场景中是否激活。
     *   @return     在场景中是否激活。
     */
    get activeInHierarchy(): boolean {
        return this._getBit(NodeFlags.ACTIVE_INHIERARCHY);
    }
    /**
     * @private
     */
    protected _onActive(): void {
        Stat.spriteCount++;
    }
    /**
     * @private
     */
    protected _onInActive(): void {
        Stat.spriteCount--;
    }
    /**
     * @private
     */
    protected _onActiveInScene(): void {
        this.event(Node.EVENT_SET_ACTIVESCENE, this._scene);
        //override it.
    }
    /**
     * @private
     */
    protected _onInActiveInScene(): void {
        this.event(Node.EVENT_SET_IN_ACTIVESCENE, this._scene);
        //override it.
    }
    /**
    * 组件被激活后执行，此时所有节点和组件均已创建完毕，次方法只执行一次
    * 此方法为虚方法，使用时重写覆盖即可
    */
    onAwake(): void {
```

```
    //this.name  && trace("onAwake node ", this.name);
}
/**
 * 组件被启用后执行，比如节点被添加到舞台后
 * 此方法为虚方法，使用时重写覆盖即可
 */
onEnable(): void {
    //this.name  && trace("onEnable node ", this.name);
}
/**
 * 组件被禁用时执行，比如从节点从舞台移除后
 * 此方法为虚方法，使用时重写覆盖即可
 */
onDisable(): void {
    //trace("onDisable node", this.name);
}
/**
 * @internal
 */
_parse(data: any, spriteMap: any): void {
    //override it.
}
/**
 * @internal
 */
_setBelongScene(scene: Node): void {
    if (!this._scene || this.scene != scene) {
        this._scene = scene;
        this._onActiveInScene();
        for (let i = 0, n = this._children.length; i < n; i++)
            this._children[i]._setBelongScene(scene);
    }
}
/**
 * @internal
 */
_setUnBelongScene(): void {
    if (this._scene !== this) {//移除节点本身是 scene 不继续派发
        this._onInActiveInScene();
        this._scene = null;
        for (let i = 0, n = this._children.length; i < n; i++)
            this._children[i]._setUnBelongScene();
    }
}
_processActive(active: boolean, fromSetter?: boolean) {
    (this._activeChangeScripts) || (this._activeChangeScripts = []);
    let arr = this._activeChangeScripts;
    if (active)
        this._activeHierarchy(arr, fromSetter);
    else
```

```
            this._inActiveHierarchy(arr, fromSetter);
        for (let i = 0, n = arr.length; i < n; i++) {
            let comp = arr[i];
            comp.owner && comp._setActive(active);
        }
        arr.length = 0;
    }
    /**
     * @internal
     */
    _activeHierarchy(activeChangeScripts: any[], fromSetter?: boolean): void {
        this._setBit(NodeFlags.ACTIVE_INHIERARCHY, true);
        if (this._components) {
            for (let i = 0, n = this._components.length; i < n; i++) {
                let comp = this._components[i];
                if (comp._isScript())
                    (comp._enabled) && (activeChangeScripts.push(comp));
                else
                    comp._setActive(true);
            }
        }
        this._onActive();
        for (let i = 0, n = this._children.length; i < n; i++) {
            let child = this._children[i];
            (!child._getBit(NodeFlags.NOT_ACTIVE) && !child._getBit(NodeFlags.NOT_READY)) &&
(child._activeHierarchy(activeChangeScripts, fromSetter));
        }
        if (!this._getBit(NodeFlags.AWAKED)) {
            this._setBit(NodeFlags.AWAKED, true);
            this.onAwake();
        }
        this.onEnable();
    }
    /**
     * @internal
     */
    _inActiveHierarchy(activeChangeScripts: any[], fromSetter?: boolean): void {
        this._onInActive();
        if (this._components) {
            for (let i = 0, n = this._components.length; i < n; i++) {
                let comp = this._components[i];
                if (comp._isScript())
                    comp._enabled && (activeChangeScripts.push(comp));
                else
                    comp._setActive(false);
            }
        }
        this._setBit(NodeFlags.ACTIVE_INHIERARCHY, false);
        for (let i = 0, n = this._children.length; i < n; i++) {
            let child = this._children[i];
            (child && !child._getBit(NodeFlags.NOT_ACTIVE)) && (child._inActiveHierarchy(activeChangeScripts, fromSetter));
        }
        this.onDisable();
    }
    /**
     * @private
     */
    protected _onAdded(): void {
        if (this._activeChangeScripts && this._activeChangeScripts.length !== 0) {
            throw "Node: can't set the main inActive node active in hierarchy,if the operate is in main inActive node or it's children script's onDisable Event.";
        } else {
            let parentScene = this._parent.scene;
            parentScene && this._setBelongScene(parentScene);
            (this._parent.activeInHierarchy && this.active) && this._processActive(true);
        }
    }
    /**
     * @private
     */
    protected _onRemoved(): void {
        if (this._activeChangeScripts && this._activeChangeScripts.length !== 0) {
            throw "Node: can't set the main active node inActive in hierarchy,if the operate is in main active node or it's children script's onEnable Event.";
        } else {
            (this._parent.activeInHierarchy && this.active) && this._processActive(false);
            this._parent.scene && this._setUnBelongScene();
        }
    }
    /**
     * @internal
     */
    _addComponentInstance(comp: Component): void {
        if (!this._components)
            this._components = [];
        this._components.push(comp);
        comp._setOwner(this);
        if (this.activeInHierarchy)
            comp._setActive(true);
        this._componentsChanged?.(comp, 0);
    }
    /**
     * @internal
     */
    _destroyComponent(comp: Component) {
        if (!this._components)
            return;
        let i = this._components.indexOf(comp);
        if (i != -1) {
            this._components.splice(i, 1);
            comp._destroy();
            this._componentsChanged?.(comp, 1);
        }
    }
    /**
     * @internal
     */
    private destroyAllComponent(): void {
        if (!this._components)
            return;
        for (let i = 0, n = this._components.length; i < n; i++) {
            let item = this._components[i];
            item && !item.destroyed && item._destroy();
        }
        this._components.length = 0;
        this._componentsChanged?.(null, 2);
    }
    /**
     * 组件列表发生改变。
     * @private
     */
    protected _componentsChanged?(comp: Component, action: 0 | 1 | 2): void;
    /**
     * @internal 克隆。
     * @param	destObject 克隆源。
     */
    _cloneTo(destObject: any, srcRoot: Node, dstRoot: Node): void {
        var destNode: Node = (<Node>destObject);
        if (this._components) {
            for (let i = 0, n = this._components.length; i < n; i++) {
                var destComponent = destNode.addComponent((this._components[i] as any).constructor);
                this._components[i]._cloneTo(destComponent);
            }
        }
    }
    /**
     * 添加组件实例。
     * @param	component 组建实例。
     * @return组件。
     */
    addComponentInstance(component: Component): Component {
        if (component.owner)
            throw "Node:the component has belong to other node.";
        if (component._singleton && this.getComponent(((<any>component)).constructor))
            console.warn("Node:the component is singleton, can't add the second one.", component);
        else
            this._addComponentInstance(component);
        return component;
    }
```

```
/**
 * 添加组件。
 * @param      componentType 组件类型。
 * @return组件。
 */
addComponent<T extends Component>(componentType: new () => T): T {
    let comp: T = Pool.createByClass(componentType);
    if (!comp) {
        throw "missing " + componentType.toString();
    }
    if (comp._singleton && this.getComponent(componentType))
        console.warn("Node:the component is singleton, can't add the second one.", comp);
    else
        this._addComponentInstance(comp);
    return comp;
}
/**
 * 获得组件实例，如果没有则返回为 null
 * @param      componentType 组建类型
 * @return返回组件
 */
getComponent<T extends Component>(componentType: new () => T): T {
    if (this._components) {
        for (let i = 0, n = this._components.length; i < n; i++) {
            let comp = this._components[i];
            if (comp instanceof componentType)
                return comp;
        }
    }
    return null;
}
/**
 * 返回所有组件实例。
 * @return 返回组件实例数组。
 */
get components(): ReadonlyArray<Component> {
    return this._components || ARRAY_EMPTY;
}
/**
 * 获得组件实例，如果没有则返回为 null
 * @param      componentType 组件类型
 * @return返回组件数组
 */
getComponents(componentType: typeof Component): Component[] {
    var arr: any[];
    if (this._components) {
        for (let i = 0, n = this._components.length; i < n; i++) {
            let comp = this._components[i];
            if (comp instanceof componentType) {
                    arr = arr || [];
                    arr.push(comp);
                }
            }
        }
        return arr;
    }
    /**
     * 获取 timer
     */
    get timer(): Timer {
        return this._scene ? this._scene.timer : ILaya.timer;
    }
    /**
     * 反序列化后会调用
     */
    onAfterDeserialize() { }
}
const _bubbleChainPool: Array<Array<Node>> = [];
export interface INodeExtra { }
/**
 * <p><code>Handler</code> 是事件处理器类。</p>
 * <p>推荐使用 Handler.create() 方法从对象池创建，减少对象创建消耗。创建的 Handler 对象不再使用后，可以使用 Handler.recover() 将其回收到对象池，回收后不要再使用此对象，否则会导致不可预料的错误。</p>
 * <p><b>注意：</b>由于鼠标事件也用本对象池，不正确的回收及调用，可能会影响鼠标事件的执行。</p>
 */
export class Handler {
    /**@private handler 对象池*/
    protected static _pool: Handler[] = [];
    /**@private */
    private static _gid: number = 1;
    /** 执行域(this)。*/
    caller: Object | null;
    /** 处理方法。*/
    method: Function | null;
    /** 参数。*/
    args: any[] | null;
    /** 表示是否只执行一次。如果为 true，回调后执行 recover()进行回收，回收后会被再利用，默认为 false 。*/
    once = false;
    /**@private */
    protected _id = 0;
    /**
     * 根据指定的属性值，创建一个 <code>Handler</code> 类的实例。
     * @param      caller 执行域。
     * @param      method 处理函数。
     * @param      args 函数参数。
     * @param      once 是否只执行一次。
     */
    constructor(caller: Object | null = null, method: Function | null = null, args: any[] | null = null, once: boolean = false) {
        this.setTo(caller, method, args, once);
    }
    /**
     * 设置此对象的指定属性值。
     * @param      caller 执行域(this)。
     * @param      method 回调方法。
     * @param      args 携带的参数。
     * @param      once 是否只执行一次，如果为 true，执行后执行 recover()进行回收。
     * @return  返回 handler 本身。
     */
    setTo(caller: any, method: Function | null, args: any[] | null, once = false): Handler {
        this._id = Handler._gid++;
        this.caller = caller;
        this.method = method;
        this.args = args;
        this.once = once;
        return this;
    }
    /**
     * 执行处理器。
     */
    run(): any {
        if (this.method == null) return null;
        var id: number = this._id;
        var result: any = this.method.apply(this.caller, this.args);
        this._id === id && this.once && this.recover();
        return result;
    }
    /**
     * 执行处理器，并携带额外数据。
     * @param      data 附加的回调数据，可以是单数据或者 Array(作为多参)。
     */
    runWith(data: any): any {
        if (this.method == null) return null;
        var id: number = this._id;
        if (data == null)
            var result: any = this.method.apply(this.caller, this.args);
        else if (!this.args && !data.unshift) result = this.method.call(this.caller, data);
        else if (this.args) result = this.method.apply(this.caller, this.args.concat(data));
        else result = this.method.apply(this.caller, data);
        this._id === id && this.once && this.recover();
        return result;
    }
    /**
     * 清理对象引用。
     */
    clear(): Handler {
        this.caller = null;
        this.method = null;
        this.args = null;
        return this;
    }
    /**
     * 清理并回收到 Handler 对象池内。
     */
    recover(): void {
        if (this._id > 0) {
            this._id = 0;
            Handler._pool.push(this.clear());
        }
    }
    /**
     * 从对象池内创建一个 Handler，默认会执行一次并立即回收，如果不需要自动回收，设置 once 参数为 false。
     * @param    caller 执行域(this)。
     * @param    method 回调方法。
     * @param    args 携带的参数。
     * @param    once 是否只执行一次，如果为 true，回调后执行 recover()进行回收，默认为 true。
     * @return  返回创建的 handler 实例。
     */
    static create(caller: any, method: Function | null, args: any[] | null = null, once: boolean = true): Handler {
        if (Handler._pool.length)
            return (Handler._pool.pop() as Handler).setTo(caller, method, args, once);
        return new Handler(caller, method, args, once);
    }
}
import { ConfigPath } from "../const/ConfigPath";
import { ViewMgr } from "../core/UI/ViewMgr";
import { Game } from "../views/Game";
import { EViewKey } from "../views/ViewConst";
import { ELevelConst } from "../views/level/LevelConst";
import { LevelModel } from "../views/level/LevelModel";
import { BackgroundRoot } from "./BackgroundRoot";
import { InputManager } from "./InputManager";
import { EItemType } from "./Item/EItemType";
import { LevelCamera } from "./LevelCamera";
import { Player } from "./Player";
import { LevelNodeManager } from "./levelParse/LevelNodeManager";
/**
 * author: 陈秀齐
 * time: 2023/12/14 11:05:26
 * desc:
 */
const { regClass, property } = Laya;
@regClass('81d36ae9-c41d-47cc-b112-cc4568ccd384', '../src/level/Level.ts')
export class Level extends Laya.Script {
    declare owner: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "地面根节点" })
    private moveRoot: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "地面根节点" })
    private groundRoot: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "障碍物根节点" })
    private obstacleRoot: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "物品根节点" })
    private itemRoot: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "特效根节点" })
    private effectRoot: Laya.Sprite;
    @property({ type: Laya.Sprite, tips: "ui 根节点" })
    private uiRoot: Laya.Sprite;
    @property({ type: InputManager, tips: "关卡输入控制" })
    private inputManager: InputManager;
    @property({ type: LevelNodeManager, tips: "关卡节点管理" })
    private nodeManager: LevelNodeManager;
    @property({ type: Player, tips: "角色" })
    private player: Player;
    @property({ type: LevelCamera, tips: "相机控制" })
    public levelCamera: LevelCamera;
    @property({ type: ["Record", Number], tips: "配置" })
    public config: Record<string, number>;
    @property({ type: Laya.Animation, tips: "受击特效" })
    private animHurt: Laya.Animation;
    @property({ type: Laya.Animation, tips: "落地特效" })
    private animDust: Laya.Animation;
    public backgroundRoot: BackgroundRoot;
    private _isInit: boolean = false;
    private _enabledCollision: boolean = true;
    get spawnPoint(): [number, number] {
        const { x, y } = this.player.owner;
        return [x, y];
    }
    private parsePrefabData(levelId: ELevelConst, offset: number): void {
        this.nodeManager.init(levelId, offset);
    }
    private checkCollision(): void {
        const playerRect = this.player.collisionBox;
        //
        let items = this.nodeManager.items;
        let item = this.tryCheckCollision(playerRect, items);
        // trigger
        if (item && item.type == EItemType.FinalAward) {
            Game.ins.win();
            this._enabledCollision = false;
            this.player.stop();
            this.player.pause();
        };
        if (item && item.type == EItemType.FoCat) {
            item.collisionEvent();
        };
        // collision
        let obstacles = this.nodeManager.obstacles;
        let obstacle = this.tryCheckCollision(playerRect, obstacles);
        if (obstacle) {
            this.inputManager.cancel();
            this.player.addForce(obstacle.force, obstacle.degrees);
            this.showHurtEffect();
            Laya.SoundManager.playSound(ConfigPath.M_CatHurt);
            return;
        };
        let grounds = this.nodeManager.grounds;
        let scrollGrounds = grounds.filter(g => g.isScrollBack);
        let staticGrounds = grounds.filter(g => !g.isScrollBack);
        let ground = this.tryCheckCollision(playerRect, scrollGrounds);
        if (!ground) {
            ground = this.tryCheckCollision(playerRect, staticGrounds);
        }
        const newIsGround = ground != null;
        const lastIsGround = this.player.isGround;
        if (lastIsGround != newIsGround) {
            this.player.isGround = newIsGround;
            // 落地
            if (newIsGround) {
                if (ground.isScrollBack) {
                    this.player.velocityY = 0;
                    this.player.velocityX = -ground.moveSpeed;
                } else {
                    this.player.stop();
                }
                this.recordPlayerPos();
                this.showDustEffect();
                this.player.owner.y = ground.owner.y;
                Laya.SoundManager.playSound(ConfigPath.M_Foot);
            }
        }
    }
    private tryCheckCollision<T extends { collisionBox: Laya.Rectangle }>(playerRect: Laya.Rectangle, list: T[]): T {
        return list.find(i => i.collisionBox.intersects(playerRect));
    }
    private onAnimDustComplete(): void {
```

```
    this.animDust.visible = false;
}
private onAnimHurtComplete(): void {
    this.animHurt.visible = false;
}
onAwake(): void {
    this.animDust.on(Laya.Event.COMPLETE, this, this.onAnimDustComplete);
    this.animHurt.on(Laya.Event.COMPLETE, this, this.onAnimHurtComplete);
}
onUpdate(): void {
    if (!this._isInit) return;
    if (this._enabledCollision) {
        this.checkCollision();
    }
}
onDestroy(): void {
    LevelModel.ins.resetDistance();
}
init(levelId: number, backgroundRoot: BackgroundRoot): void {
    if (this._isInit) return;
    this._isInit = true;
    // levelId = ELevelConst.Level_10002;
    LevelModel.ins.currId = levelId;
    let startLine = this.config[levelId];
    this.moveRoot.x = startLine;
    this.uiRoot.x = -startLine;
    let realStartPos = startLine - this.spawnPoint[0];
    LevelModel.ins.setStartSpace(realStartPos);
    this.parsePrefabData(levelId, startLine);
    this.player.spawn(...this.spawnPoint);
    this.inputManager.init(this.player);
    this.levelCamera.init(this.player);
    this.levelCamera.addFollower(backgroundRoot);
    this.backgroundRoot = backgroundRoot;
}
recordPlayerPos(): void {
    LevelModel.ins.recordPlayerPos(this.levelCamera.distance);
}
reEnterLevel(levelId: number): void {
    LevelModel.ins.currId = levelId;
    this.levelCamera.backToStart();
    LevelModel.ins.resetDistance();
    let startLine = this.config[levelId];
    this.moveRoot.x = startLine;
    this.uiRoot.x = -startLine;
    let realStartPos = startLine - this.spawnPoint[0];
    LevelModel.ins.setStartSpace(realStartPos);
    this.nodeManager.clear();
    this.parsePrefabData(levelId, startLine);
    this._enabledCollision = true;
    this.player.resume();
    }
    restart(): void {
        this.reEnterLevel(LevelModel.ins.currId);
    }
    scrollTo(pos: number): void {
        this.player.hide();
        this._enabledCollision = false;
        this.inputManager.enabled = false;
        LevelModel.ins.isScrollClose = true;
        ViewMgr.ins.close(EViewKey.HudView);
        this.levelCamera.scrollTo(pos, Laya.Handler.create(this, () => {
            this.player.show();
            this._enabledCollision = true;
            this.inputManager.enabled = true;
            LevelModel.ins.scrollEnd();
            ViewMgr.ins.open(EViewKey.HudView);
            LevelModel.ins.isScrollClose = false;
        }, null, true));
    }
    showDustEffect(): void {
        this.animDust.visible = true;
        let point = this.player.getFootPoint(this.effectRoot);
        this.animDust.pos(point.x, point.y);
        this.animDust.play(0, false);
    }
    showHurtEffect(): void {
        this.animHurt.visible = true;
        let point = this.player.getFootPoint(this.effectRoot);
        this.animHurt.pos(point.x, point.y);
        this.animHurt.play(0, false);
    }
}
/**
 * author: 陈秀齐
 * time: 2023/12/12 15:22:31
 * desc: 背景视差移动
 * MTC
 * todo:
 * 1.屏幕宽度适应问题
 */
const { regClass, property } = Laya;
@regClass()
export class Background extends Laya.Script {
    declare owner: Laya.Image;
    @property({ type: Number, tips: "移动视差比例" })
    moveScale: number = 1;
    @property({ type: Number, tips: "纹理默认宽度" })
    textureWidth: number = 720;
    @property({ type: Number, tips: "初始宽度为原始宽度的倍数" })
    repeatX: number = 3;
    private _startPosX: number = 0;
    private get distance(): number {
        return Math.abs(this.owner.x - this._startPosX);
    }
    private resetPos(): void {
        let real = this.owner.x - this._startPosX;
        this.owner.x = this._startPosX + real % this.textureWidth;
    }
    private isOutOfBounds(): boolean {
        return this.distance * this.repeatX > this.textureWidth;
    }
    onStart(): void {
        const stageW = this.textureWidth * this.repeatX;
        this.owner.x = -stageW;
        this.owner.width = stageW * this.repeatX;
        this._startPosX = this.owner.x;
    }
    move(distance: number): void {
        this.owner.x += distance * this.moveScale;
        if (this.isOutOfBounds()) {
            this.resetPos();
        }
    }
}
import { Background } from "./Background";
import { CameraFollower } from "./CameraFollower";
/**
 * author: 陈秀齐
 * time: 2023/12/12 21:00:37
 * desc:
 */
const { regClass, property } = Laya;
@regClass()
export class BackgroundRoot extends CameraFollower {
    declare owner: Laya.Sprite;
    @property({ type: [Background], tips: "视差背景图层集合" })
    backgrounds: Background[] = [];
    @property({ type: Boolean, tips: "是否自动移动" })
    autoMove: boolean = false;
    @property({ type: Number, tips: "自动移动的速度" })
    autoMoveSpeed: number = 0;
    move(distance: number): void {
        if (distance == 0) return;
        this.backgrounds.forEach(bg => bg.move(distance));
    }
    onUpdate(): void {
        if (this.autoMove) {
            this.move(this.autoMoveSpeed);
        }
    }
    randomSkin(): void {
        this.setSkin(Math.floor(Math.random() * 3));
    }
    setSkin(index: number): void {
        this.backgrounds.forEach((bg, i) => {
            bg.owner.skin = `resources/scene/bg${index}/Layer_${i}.png`;
        });
    }
    enterAnim(): void {
        Laya.Tween.from(this.owner, { alpha: 0 }, 1000)
    }
    exitAnim(): void {
        Laya.Tween.from(this.owner, { alpha: 0 }, 1000)
    }
}
import { CameraFollower } from "./CameraFollower";
/**
 * author: 陈秀齐
 * time: 2023/12/13 08:48:23
 * desc:
 */
export interface ICameraFocusTarget {
    velocityX: number;
}
```