0001: import { LevelLoader } from "./LevelLoader";

0002: import { Game } from "./views/Game";

0003: import { LoadingViewRT } from "./views/loading/LoadingViewRT";

0004: const { regClass, property } = Laya;

0005: @regClass()

0006: export class Boot extends Laya.Script {

0007: declare owner: Laya.Sprite;

0008: @property({ type: Laya.Prefab })

0009: private loadingPrefab: Laya.Prefab;

0010: @property({ type: LevelLoader })

0011: public levelLoader: LevelLoader;

0012: private \_loadingNode: LoadingViewRT;

0013: //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

0014: onAwake(): void {

0015: let node = this.loadingPrefab.create() as LoadingViewRT;

0016: this.\_loadingNode = node;

0017: node.value = 0;

0018: Laya.Scene.setLoadingPage(node);

0019: Laya.Scene.showLoadingPage();

0020: Game.ins.init(this);

0021: }

0022: setLoading(data: { desc?: string, value?: number }): void {

0023: data.desc != null && (this.\_loadingNode.desc = data.desc);

0024: data.value != null && (this.\_loadingNode.value = data.value);

0025: }

0026: }

0027: import { t, StateMachine } from "./core/FSM/StateMatchine";

0028: export enum GameStates {

0029: init = "init",

0030: loading = "loading",

0031: loaded = "loaded",

0032: home = "home",

0033: level = "level",

0034: pause = "pause",

0035: win = "win",

0036: }

0037: export enum GameEvents {

0038: load = 100,

0039: loadComplete,

0040: enterHome,

0041: enterLevel,

0042: win,

0043: nextLevel,

0044: pause,

0045: resume,

0046: restartLevel,

0047: backHome,

0048: }

0049: export class GameFSM extends StateMachine<GameStates, GameEvents> {

0050: private readonly \_id = `GameFSM\_${Math.floor(Math.random() \* 10000)}`;

0051: public onLoadHandler: Laya.Handler;

0052: public onLoadCompleteHandler: Laya.Handler;

0053: public onEnterHomeHandler: Laya.Handler;

0054: public onEnterLevelHandler: Laya.Handler;

0055: public onWinHandler: Laya.Handler;

0056: public onPauseHandler: Laya.Handler;

0057: public onNextLevelHandler: Laya.Handler;

0058: public onResumeHandler: Laya.Handler;

0059: public onRestartLevelHandler: Laya.Handler;

0060: public onBackHomeHandler: Laya.Handler;

0061: constructor(init = GameStates.init) {

0062: super(init);

0063: const s = GameStates;

0064: const e = GameEvents;

0065: this.addTransitions([

0066: t(s.init, e.load, s.loading, this.onLoad),

0067: t(s.loading, e.loadComplete, s.loaded, this.onLoadComplete),

0068: t(s.loaded, e.enterHome, s.home, this.onEnterHome),

0069: t(s.home, e.enterLevel, s.level, this.onEnterLevel),

0070: t(s.level, e.win, s.win, this.onWin),

0071: t(s.level, e.pause, s.pause, this.onPause),

0072: t(s.win, e.nextLevel, s.level, this.onNextLevel),

0073: t(s.win, e.restartLevel, s.level, this.onRestartLevel),

0074: t(s.win, e.backHome, s.home, this.onBackHome),

0075: t(s.pause, e.resume, s.level, this.onResume),

0076: t(s.pause, e.restartLevel, s.level, this.onRestartLevel),

0077: t(s.pause, e.backHome, s.home, this.onBackHome),

0078: ])

0079: }

0080: private logState(): void {

0081: this.logger.log(`${this.\_id} ${[GameStates[this.getState()]]}`);

0082: }

0083: private async onLoad(): Promise<void> {

0084: this.onLoadHandler && this.onLoadHandler.run();

0085: }

0086: private async onEnterHome(): Promise<void> {

0087: this.onEnterHomeHandler && this.onEnterHomeHandler.run();

0088: }

0089: private async onLoadComplete(): Promise<void> {

0090: this.onLoadCompleteHandler && this.onLoadCompleteHandler.run();

0091: }

0092: private async onEnterLevel(levelId: number): Promise<void> {

0093: this.onEnterLevelHandler && this.onEnterLevelHandler.runWith(levelId);

0094: }

0095: private async onWin(): Promise<void> {

0096: this.onWinHandler && this.onWinHandler.run();

0097: }

0098: private async onPause(): Promise<void> {

0099: this.onPauseHandler && this.onPauseHandler.run();

0100: }

0101: private async onNextLevel(levelId: number): Promise<void> {

0102: this.onNextLevelHandler && this.onNextLevelHandler.runWith(levelId);

0103: }

0104: private async onResume(): Promise<void> {

0105: this.onResumeHandler && this.onResumeHandler.run();

0106: }

0107: private async onRestartLevel(): Promise<void> {

0108: this.onRestartLevelHandler && this.onRestartLevelHandler.run();

0109: }

0110: private async onBackHome(): Promise<void> {

0111: this.onBackHomeHandler && this.onBackHomeHandler.run();

0112: }

0113: }

0114: import { SceneRegUtils } from "./core/UI/SceneRegUtils";

0115: import { BackgroundRoot } from "./level/BackgroundRoot";

0116: import { Level } from "./level/Level";

0117: import { LevelLoadMask } from "./level/LevelLoadMask";

0118: import { EViewLayer } from "./views/ViewConst";

0119: import { LevelModel } from "./views/level/LevelModel";

0120: /\*\*

0121: \* author: 陈秀齐

0122: \* time: 2023/12/13 16:11:00

0123: \* desc:

0124: \*/

0125: const { regClass, property } = Laya;

0126: @regClass()

0127: export class LevelLoader extends Laya.Script {

0128: @property({ type: Laya.Prefab, tips: "关卡基础控件" })

0129: levelBasePrefab: Laya.Prefab;

0130: @property({ type: Laya.Prefab, tips: "视差滚动背景" })

0131: backgroundRootPrefab: Laya.Prefab;

0132: @property({ type: Laya.Prefab, tips: "场景过度动画节点" })

0133: loadMask: Laya.Prefab;

0134: get root(): Laya.Sprite {

0135: return this.owner.parent as Laya.Sprite;

0136: }

0137: createLoadMask(): LevelLoadMask {

0138: let node = this.loadMask.create();

0139: SceneRegUtils.tryAddChild(EViewLayer.UILoading, node as Laya.Sprite);

0140: return node.getComponent(LevelLoadMask);

0141: }

0142: createBackgroundRoot(): BackgroundRoot {

0143: let node = this.backgroundRootPrefab.create();

0144: this.root.addChild(node);

0145: return node.getComponent(BackgroundRoot);

0146: }

0147: loadLevel(levelId: number, backgroundRoot: BackgroundRoot): Level {

0148: let levelNode = this.levelBasePrefab.create();

0149: this.root.addChild(levelNode);

0150: let level = levelNode.getComponent(Level);

0151: level.init(levelId, backgroundRoot);

0152: return level;

0153: }

0154: unloadLevel(level: Level): void {

0155: LevelModel.ins.currId = null;

0156: level.owner.destroy();

0157: }

0158: }

0159: /\*\*

0160: \* author: 陈秀齐

0161: \* time: 2023/12/14 19:19:50

0162: \* desc:

0163: \*/

0164: import { Boot } from "../Boot";

0165: import { GameEvents, GameFSM, GameStates } from "../GameFSM";

0166: import { ConfigPath } from "../const/ConfigPath";

0167: import { SceneRegUtils } from "../core/UI/SceneRegUtils";

0168: import { ViewMgr } from "../core/UI/ViewMgr";

0169: import { ViewRegUtils } from "../core/UI/ViewRegUtils";

0170: import { Singleton } from "../core/base/Singleton";

0171: import { BackgroundRoot } from "../level/BackgroundRoot";

0172: import { Level } from "../level/Level";

0173: import { LevelLoadMask } from "../level/LevelLoadMask";

0174: import { PromiseEx } from "../utils/PromiseEx";

0175: import { ViewLayerZOrder, EViewKey, EViewLayer } from "./ViewConst";

0176: import { ELevelConst } from "./level/LevelConst";

0177: import { LevelModel } from "./level/LevelModel";

0178: export class Game extends Singleton<Game>() {

0179: private \_fsm: GameFSM;

0180: private \_boot: Boot;

0181: private \_level: Level;

0182: private \_levelLoadMask: LevelLoadMask;

0183: private \_backgroundRoot: BackgroundRoot;

0184: init(boot: Boot): void {

0185: this.\_boot = boot;

0186: this.\_fsm = new GameFSM();

0187: this.\_fsm.onLoadHandler = new Laya.Handler(this, this.onLoadHandler, null, false);

0188: this.\_fsm.onLoadCompleteHandler = new Laya.Handler(this, this.onLoadCompleteHandler, null, false);

0189: this.\_fsm.onEnterHomeHandler = new Laya.Handler(this, this.onEnterHomeHandler, null, false);

0190: this.\_fsm.onEnterLevelHandler = new Laya.Handler(this, this.onEnterLevelHandler, null, false);

0191: this.\_fsm.onWinHandler = new Laya.Handler(this, this.onWinHandler, null, false);

0192: this.\_fsm.onPauseHandler = new Laya.Handler(this, this.onPauseHandler, null, false);

0193: this.\_fsm.onNextLevelHandler = new Laya.Handler(this, this.onNextLevelHandler, null, false);

0194: this.\_fsm.onResumeHandler = new Laya.Handler(this, this.onResumeHandler, null, false);

0195: this.\_fsm.onRestartLevelHandler = new Laya.Handler(this, this.onRestartLevelHandler, null, false);

0196: this.\_fsm.onBackHomeHandler = new Laya.Handler(this, this.onBackHomeHandler, null, false);

0197: this.\_fsm.dispatch(GameEvents.load);

0198: }

0199: private async sequeueInit() {

0200: // 设置语言包

0201: await this.initLangPacks();

0202: this.\_boot.setLoading({ desc: "100003", value: 0.05 });

0203: await PromiseEx.delay(20);

0204: // 初始化场景层级

0205: this.buildScene();

0206: this.\_boot.setLoading({ desc: "100001", value: 0.10 });

0207: await PromiseEx.delay(20);

0208: // 初始化场景注册信息

0209: this.registerAllView();

0210: this.\_boot.setLoading({ desc: "100002", value: 0.15 });

0211: await PromiseEx.delay(20);

0212: // 设置加载Common资源

0213: this.\_boot.setLoading({ desc: "100004" });

0214: this.loadRes();

0215: Laya.SoundManager.setMusicVolume(0.5);

0216: }

0217: // 设置语言包

0218: private async initLangPacks(): Promise<any> {

0219: return Laya.loader.load(ConfigPath.JSON\_Lang).then((result) => {

0220: Laya.Text.langPacks = result.data;

0221: })

0222: }

0223: // 初始化场景层级

0224: private buildScene(): void {

0225: let orders = ViewLayerZOrder;

0226: orders.forEach((item) => {

0227: const [layer, zOrder] = item;

0228: SceneRegUtils.add(layer, new Laya.Sprite(), zOrder);

0229: });

0230: }

0231: // 初始化场景注册信息

0232: private registerAllView(): void {

0233: const k = EViewKey, l = EViewLayer;

0234: ViewRegUtils.register(k.MainView, l.UI, { showMask: false, extraClick: false, enterAnim: false }, ConfigPath.LH\_MainView);

0235: ViewRegUtils.register(k.SkinView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH\_SkinView);

0236: ViewRegUtils.register(k.HelpView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH\_Help);

0237: ViewRegUtils.register(k.HudView, l.UI, { showMask: false, extraClick: false, enterAnim: false }, ConfigPath.LH\_Hud);

0238: ViewRegUtils.register(k.PauseView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH\_PauseView);

0239: ViewRegUtils.register(k.WinView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH\_WinView);

0240: ViewRegUtils.register(k.WinGoldView, l.UI, { showMask: true, extraClick: false, enterAnim: false }, ConfigPath.LH\_WinGoldView);

0241: }

0242: private loadRes(): void {

0243: Laya.loader.on(Laya.Event.ERROR, this, this.onLoadError);

0244: Laya.loader.load(ConfigPath.EnterLoadList, null, Laya.Handler.create(this, this.onLoadProgress)).then(() => {

0245: this.onLoadCompleted();

0246: })

0247: }

0248: private openMainView(): void {

0249: this.\_backgroundRoot.setSkin(0);

0250: this.\_levelLoadMask.ungroup(Laya.Handler.create(this, () => {

0251: ViewMgr.ins.open(EViewKey.MainView);

0252: Laya.SoundManager.playMusic(ConfigPath.M\_Main);

0253: }));

0254: }

0255: private onLoadCompleted(): void {

0256: this.\_boot.setLoading({ value: 0.95 });

0257: PromiseEx.delay(20).then(() => {

0258: this.\_boot.setLoading({ desc: "100005", value: 1.0 });

0259: Laya.Scene.hideLoadingPage(0);

0260: this.\_levelLoadMask = this.\_boot.levelLoader.createLoadMask();

0261: this.\_backgroundRoot = this.\_boot.levelLoader.createBackgroundRoot();

0262: this.\_backgroundRoot.enterAnim();

0263: Laya.timer.once(200, this, () => {

0264: this.\_fsm.dispatch(GameEvents.loadComplete);

0265: })

0266: });

0267: }

0268: private onLoadProgress(progress: number): void {

0269: console.log("progress=====", progress);

0270: }

0271: private onLoadError(error: string): void {

0272: console.log("error=====", error);

0273: }

0274: private onLoadHandler(): void {

0275: this.sequeueInit();

0276: }

0277: private onLoadCompleteHandler(): void {

0278: this.\_fsm.dispatch(GameEvents.enterHome);

0279: }

0280: private onEnterHomeHandler(): void {

0281: this.openMainView();

0282: }

0283: private onEnterLevelHandler(levelId: number): void {

0284: this.\_levelLoadMask.group(Laya.Handler.create(this, this.onEnterLevelGroupHandler, [levelId]));

0285: }

0286: private onEnterLevelGroupHandler(levelId: number): void {

0287: ViewMgr.ins.close(EViewKey.MainView);

0288: this.\_backgroundRoot.autoMove = false;

0289: this.\_backgroundRoot.setSkin(LevelModel.ins.skin);

0290: this.\_level = this.\_boot.levelLoader.loadLevel(levelId, this.\_backgroundRoot);

0291: ViewMgr.ins.open(EViewKey.HudView);

0292: this.\_levelLoadMask.ungroup();

0293: // Laya.SoundManager.playMusic(ConfigPath.M\_Level);

0294: }

0295: private onWinHandler(): void {

0296: ViewMgr.ins.close(EViewKey.HudView);

0297: if (LevelModel.ins.isSecondLevel()) {

0298: ViewMgr.ins.open(EViewKey.WinGoldView);

0299: } else {

0300: ViewMgr.ins.open(EViewKey.WinView);

0301: }

0302: }

0303: private onPauseHandler(): void {

0304: ViewMgr.ins.close(EViewKey.HudView);

0305: ViewMgr.ins.open(EViewKey.PauseView);

0306: }

0307: private onNextLevelHandler(): void {

0308: this.\_levelLoadMask.group(Laya.Handler.create(this, this.onNextLevelGroupHandler));

0309: }

0310: private onNextLevelGroupHandler(): void {

0311: const levelId = ELevelConst.Level\_10002;

0312: this.\_level.reEnterLevel(levelId);

0313: ViewMgr.ins.close(EViewKey.WinView);

0314: ViewMgr.ins.open(EViewKey.HudView);

0315: this.\_levelLoadMask.ungroup();

0316: // Laya.SoundManager.playMusic(ConfigPath.M\_Level);

0317: }

0318: private onResumeHandler(): void {

0319: ViewMgr.ins.open(EViewKey.HudView);

0320: ViewMgr.ins.close(EViewKey.PauseView);

0321: }

0322: private onRestartLevelHandler(): void {

0323: this.\_levelLoadMask.group(Laya.Handler.create(this, this.onRestartLevelGroupHandler));

0324: }

0325: private onRestartLevelGroupHandler(): void {

0326: this.\_levelLoadMask.ungroup();

0327: ViewMgr.ins.close(EViewKey.PauseView);

0328: ViewMgr.ins.close(EViewKey.WinGoldView);

0329: this.\_level.restart();

0330: ViewMgr.ins.open(EViewKey.HudView);

0331: // Laya.SoundManager.playMusic(ConfigPath.M\_Level);

0332: }

0333: private onBackHomeHandler(): void {

0334: this.\_levelLoadMask.group(Laya.Handler.create(this, this.onBackHomeGroupHandler));

0335: }

0336: private onBackHomeGroupHandler(): void {

0337: this.\_boot.levelLoader.unloadLevel(this.\_level);

0338: this.\_level = null;

0339: ViewMgr.ins.close(EViewKey.PauseView);

0340: ViewMgr.ins.close(EViewKey.WinGoldView);

0341: this.openMainView();

0342: }

0343: enterLevel(levelId: number): void {

0344: this.\_fsm.dispatch(GameEvents.enterLevel, levelId);

0345: }

0346: pause(): void {

0347: this.\_fsm.dispatch(GameEvents.pause, "param");

0348: }

0349: resume(): void {

0350: this.\_fsm.dispatch(GameEvents.resume);

0351: }

0352: restartLevel(): void {

0353: this.\_fsm.dispatch(GameEvents.restartLevel);

0354: }

0355: backHome(): void {

0356: this.\_fsm.dispatch(GameEvents.backHome);

0357: }

0358: nextLevel(): void {

0359: this.\_fsm.dispatch(GameEvents.nextLevel);

0360: }

0361: win(): void {

0362: this.\_fsm.dispatch(GameEvents.win);

0363: }

0364: isWin(): boolean {

0365: return this.\_fsm.getState() === GameStates.win;

0366: }

0367: scrollTo(p: number): void {

0368: this.\_level.scrollTo(p);

0369: }

0370: }

0371: export enum EViewKey {

0372: HudView = "HudView",

0373: PauseView = "PauseView",

0374: HelpView = "HelpView",

0375: MainView = "MainView",

0376: SkinView = "SkinView",

0377: WinView = "WinView",

0378: WinGoldView = "WinGoldView",

0379: }

0380: export enum EViewLayer {

0381: Bg = "bg",

0382: Battle = "battle",

0383: UI = "ui\_full",

0384: UISystem = "ui\_main",

0385: UIPopup = "ui\_popup",

0386: UIMsg = "ui\_msg",

0387: UIGuide = "ui\_guide",

0388: UILoading = "ui\_loading",

0389: UIAlert = "ui\_alert",

0390: }

0391: export const ViewLayerZOrder: [EViewLayer, number][] = [

0392: /\*\*Bg层 \*/

0393: [EViewLayer.Bg, 10],

0394: /\*\*fight层 \*/

0395: [EViewLayer.Battle, 20],

0396: /\*\*UI层 \*/

0397: [EViewLayer.UI, 100],

0398: /\*\*一级窗口UI层 \*/

0399: [EViewLayer.UISystem, 200],

0400: /\*\*二级弹窗UI层 \*/

0401: [EViewLayer.UIPopup, 300],

0402: /\*\*飘字信息UI层 \*/

0403: [EViewLayer.UIMsg, 400],

0404: /\*\*引导层 \*/

0405: [EViewLayer.UIGuide, 500],

0406: /\*\*loading \*/

0407: [EViewLayer.UILoading, 600],

0408: /\*\*提示窗口层 \*/

0409: [EViewLayer.UIAlert, 700],

0410: ];

0411: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

0412: /\*\*

0413: \* resources/prefabs/views/Help.lh

0414: \*/

0415: export class HelpViewRTBase extends Laya.Box {

0416: public lblTitle!: Laya.Label;

0417: public lblDesc!: Laya.Label;

0418: public btnClose!: Laya.Button;

0419: }

0420: const { regClass } = Laya;

0421: import { ViewMgr } from "../../core/UI/ViewMgr";

0422: import { EViewKey } from "../ViewConst";

0423: import { HelpViewRTBase } from "./HelpViewRT.generated";

0424: @regClass()

0425: export class HelpViewRT extends HelpViewRTBase {

0426: private onClickClose(): void {

0427: ViewMgr.ins.close(EViewKey.HelpView);

0428: }

0429: onAwake(): void {

0430: this.btnClose.on(Laya.Event.CLICK, this.onClickClose);

0431: }

0432: }

0433: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

0434: /\*\*

0435: \* resources/prefabs/views/HUD.lh

0436: \*/

0437: export class HudViewRTBase extends Laya.Box {

0438: public btnBack!: Laya.Button;

0439: public btnScroll!: Laya.Button;

0440: public progress!: Laya.ProgressBar;

0441: public imgHead!: Laya.Image;

0442: public imgAward!: Laya.Image;

0443: public boxMask!: Laya.Box;

0444: public btnContinue!: Laya.Button;

0445: public lblDistance!: Laya.Label;

0446: }

0447: const { regClass } = Laya;

0448: import { Game } from "../Game";

0449: import { LevelEvent } from "../level/LevelConst";

0450: import { LevelModel } from "../level/LevelModel";

0451: import { SkinModel } from "../skin/SkinModel";

0452: import { HudViewRTBase } from "./HudViewRT.generated";

0453: @regClass()

0454: export class HudViewRT extends HudViewRTBase {

0455: private updateView(): void {

0456: this.updateDistance();

0457: this.updateProgress();

0458: this.updateScrollButton();

0459: }

0460: private updateProgress(): void {

0461: const isShowProgress = LevelModel.ins.isShowProgress();

0462: this.progress.visible = isShowProgress;

0463: if (isShowProgress) {

0464: this.imgHead.skin = SkinModel.ins.getCurrentSkinHead();

0465: this.progress.value = LevelModel.ins.currDistanceFormat \* 0.01;

0466: this.imgHead.x = this.progress.width \* this.progress.value;

0467: }

0468: }

0469: private updateDistance(): void {

0470: const isPractice = LevelModel.ins.isPracticeMode();

0471: this.lblDistance.visible = isPractice;

0472: if (isPractice) {

0473: const distance = LevelModel.ins.currDistanceFormat;

0474: this.lblDistance.text = `当前距离：${distance}`;

0475: }

0476: }

0477: private updateScrollButton(): void {

0478: this.btnScroll.visible = LevelModel.ins.isExistScrollButton();

0479: }

0480: private onClickBack(): void {

0481: Game.ins.pause();

0482: }

0483: private onClickScroll(): void {

0484: Game.ins.scrollTo(LevelModel.ins.currTopDistance);

0485: }

0486: private onClickContinue(): void {

0487: this.btnBack.visible = true;

0488: this.boxMask.visible = false;

0489: }

0490: onAwake(): void {

0491: this.btnBack.on(Laya.Event.CLICK, this, this.onClickBack);

0492: this.btnScroll.on(Laya.Event.CLICK, this, this.onClickScroll);

0493: this.btnContinue.on(Laya.Event.CLICK, this, this.onClickContinue);

0494: }

0495: onEnable(): void {

0496: if (LevelModel.ins.isScrollClose) {

0497: this.btnBack.visible = false;

0498: this.boxMask.visible = true;

0499: }

0500: this.updateView();

0501: LevelModel.ins.on(LevelEvent.DistanceChanged, this, this.updateView);

0502: }

0503: onDisable(): void {

0504: LevelModel.ins.off(LevelEvent.DistanceChanged, this, this.updateView);

0505: }

0506: }

0507: import { ConfigPath } from "../../const/ConfigPath";

0508: export interface ILevelPrefabData {

0509: name: string,

0510: visible: boolean,

0511: x: number,

0512: y: number,

0513: width: number,

0514: height: number,

0515: anchorX: number,

0516: anchorY: number,

0517: \_$prefab: string

0518: }

0519: export class LevelEvent {

0520: static readonly DistanceChanged = "DistanceChanged";

0521: static readonly TopDistanceChanged = "TopDistanceChanged";

0522: }

0523: export enum ELevelMode {

0524: Practice,

0525: Normal,

0526: }

0527: export enum ELevelConst {

0528: LevelTestId = 0,

0529: LevelPracticeId = 10000,

0530: Level\_10001 = 10001,

0531: Level\_10002 = 10002,

0532: }

0533: export const LevelConfig = {

0534: [ELevelConst.LevelTestId]: { path: ConfigPath.LH\_Level\_Test },

0535: [ELevelConst.LevelPracticeId]: { path: ConfigPath.LH\_Level\_10000 },

0536: [ELevelConst.Level\_10001]: { path: ConfigPath.LH\_Level\_10001 },

0537: [ELevelConst.Level\_10002]: { path: ConfigPath.LH\_Level\_10002 },

0538: }

0539: export enum ELevelNodeSign {

0540: Item = "itemRoot",

0541: Ground = "groundRoot",

0542: Obstacle = "obstacleRoot",

0543: }

0544: export type ILevelParseProp<T extends Laya.Component> = {

0545: name: string;

0546: root: Laya.Sprite;

0547: components: T[],

0548: component: new () => T;

0549: }

0550: /\*\*

0551: \* author: 陈秀齐

0552: \* time: 2023/12/19 15:31:20

0553: \* desc:

0554: \*/

0555: import { LocalData } from "../../utils/LocalData";

0556: export type ILevelLocalData = { [key: number]: { topScore: number } }

0557: export class LevelLocalData extends LocalData<ILevelLocalData> {

0558: static readonly Key = "LevelLocalData";

0559: static readonly Default: ILevelLocalData = {};

0560: constructor() {

0561: super(LevelLocalData.Key, LevelLocalData.Default);

0562: }

0563: newHistoryRecord(levelId: number, score: number): void {

0564: if (!this.data[levelId]) {

0565: this.data[levelId] = { topScore: score };

0566: } else {

0567: this.data[levelId].topScore = score;

0568: }

0569: this.save();

0570: }

0571: getHistoryRecord(levelId: number): number {

0572: return this.data[levelId] ? this.data[levelId].topScore : 0;

0573: }

0574: }

0575: /\*\*

0576: \* author: 陈秀齐

0577: \* time: 2023/12/19 15:28:18

0578: \* desc:

0579: \*/

0580: import { Model } from "../../core/mvc/Model";

0581: import { ConfigUtils, DialogConfigData } from "../../utils/ConfigUtils";

0582: import { ELevelConst, LevelEvent } from "./LevelConst";

0583: import { LevelLocalData } from "./LevelLocalData";

0584: export class LevelModel extends Model {

0585: private static \_ins: LevelModel;

0586: static get ins(): LevelModel {

0587: if (!this.\_ins) {

0588: this.\_ins = new LevelModel();

0589: }

0590: return this.\_ins;

0591: }

0592: private \_distanceRatio = 100 / 16000;

0593: private \_startSpace = 0;

0594: private \_currId: number;

0595: private \_freeJumpTimes: number;

0596: private \_isScrollClose: boolean;

0597: private \_localData: LevelLocalData;

0598: /\*\* 当前关卡当前距离 \*/

0599: private \_currDistance: number = 0;

0600: /\*\* 当前关卡最高距离 \*/

0601: private \_currTopDistance: number = 0;

0602: private \_dialogIndex = 0;

0603: private \_enterLevelCount = 0;

0604: private constructor() {

0605: super();

0606: this.\_localData = new LevelLocalData();

0607: }

0608: get currId(): number {

0609: return this.\_currId;

0610: }

0611: set currId(v: number) {

0612: this.\_currId = v;

0613: this.\_dialogIndex = 0;

0614: this.\_freeJumpTimes = 1;

0615: this.\_enterLevelCount++;

0616: }

0617: get skin(): number {

0618: return this.\_enterLevelCount <= 1 ? 0 : Math.floor(Math.random() \* 3);

0619: }

0620: get isScrollClose(): boolean {

0621: return this.\_isScrollClose;

0622: }

0623: set isScrollClose(v: boolean) {

0624: this.\_isScrollClose = v;

0625: }

0626: set currDistance(v: number) {

0627: this.\_currDistance = v;

0628: this.event(LevelEvent.DistanceChanged, v);

0629: if (this.\_currDistance > this.currTopDistance) {

0630: this.currTopDistance = v;

0631: this.event(LevelEvent.TopDistanceChanged, v);

0632: }

0633: }

0634: get currDistance(): number {

0635: return this.\_currDistance;

0636: }

0637: set currTopDistance(v: number) {

0638: this.\_currTopDistance = v;

0639: }

0640: get currTopDistance(): number {

0641: return this.\_currTopDistance;

0642: }

0643: get currHistoryTopDistance(): number {

0644: return this.\_localData.getHistoryRecord(this.\_currId);

0645: }

0646: get currDistanceFormat(): number {

0647: return this.formatDistance(this.currDistance);

0648: }

0649: get currTopDistanceFormat(): number {

0650: return this.formatDistance(this.currTopDistance);

0651: }

0652: get currHistoryTopDistanceFormat(): number {

0653: return this.formatDistance(this.currHistoryTopDistance);

0654: }

0655: private formatDistance(distance: number): number {

0656: const real = Math.floor(distance \* this.\_distanceRatio);

0657: return Math.max(real, 0);

0658: }

0659: setStartSpace(v: number): void {

0660: this.\_startSpace = v;

0661: }

0662: isPracticeMode(): boolean {

0663: return this.currId == ELevelConst.LevelPracticeId;

0664: }

0665: isSecondLevel(): boolean {

0666: return this.currId == ELevelConst.Level\_10002;

0667: }

0668: isShowProgress(): boolean {

0669: return this.isSecondLevel() && this.currDistance > this.\_startSpace;

0670: }

0671: isExistTop(): boolean {

0672: return this.isSecondLevel() && this.currTopDistance > this.currDistance;

0673: }

0674: isExistScrollButton(): boolean {

0675: return this.isExistFree() && (this.isPracticeMode() || this.isSecondLevel()) && this.currTopDistance > this.currDistance;

0676: }

0677: isExistFree(): boolean {

0678: return this.\_freeJumpTimes > 0;

0679: }

0680: scrollEnd(): void {

0681: this.\_freeJumpTimes--;

0682: this.recordPlayerPos(this.currTopDistance);

0683: }

0684: recordPlayerPos(distance: number): void {

0685: this.currDistance = distance;

0686: }

0687: resetDistance(): void {

0688: this.currDistance = this.currTopDistance = 0;

0689: }

0690: checkExistNewRecord(): boolean {

0691: return this.currTopDistanceFormat > this.currHistoryTopDistanceFormat;

0692: }

0693: saveNewRecord(): void {

0694: this.\_localData.newHistoryRecord(this.\_currId, this.currTopDistance);

0695: }

0696: setLableDialog(label: Laya.Label): void {

0697: const configs: DialogConfigData[] = ConfigUtils.get("dialog");

0698: const config = configs.find(c => c.id == this.\_currId);

0699: if (!config) return;

0700: let dialog = config.dialogs[this.\_dialogIndex];

0701: this.\_dialogIndex = (this.\_dialogIndex + 1) % config.dialogs.length;

0702: label.text = dialog.desc;

0703: switch (dialog.type) {

0704: case 1:

0705: label.setVar("n", this.currHistoryTopDistanceFormat);

0706: break;

0707: case 2:

0708: label.setVar("n", this.currHistoryTopDistanceFormat - this.currDistanceFormat);

0709: break;

0710: default:

0711: break;

0712: }

0713: }

0714: }

0715: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

0716: /\*\*

0717: \* resources/prefabs/views/LoadingView.lh

0718: \*/

0719: export class LoadingViewRTBase extends Laya.Box {

0720: public progress!: Laya.ProgressBar;

0721: public lblProgress!: Laya.Label;

0722: }

0723: const { regClass } = Laya;

0724: import { StringUtils } from "../../utils/StringUtils";

0725: import { LoadingViewRTBase } from "./LoadingViewRT.generated";

0726: @regClass()

0727: export class LoadingViewRT extends LoadingViewRTBase {

0728: private \_desc: string = "100000";

0729: private \_value: number;

0730: public get value(): number {

0731: return this.\_value;

0732: }

0733: public set value(v: number) {

0734: this.\_value = v;

0735: this.progress.value = v;

0736: this.lblProgress.text = `${StringUtils.lang(this.desc)}...${StringUtils.toPercent(v)}`;

0737: }

0738: public get desc(): string {

0739: return this.\_desc;

0740: }

0741: public set desc(v: string) {

0742: this.\_desc = v;

0743: }

0744: }

0745: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

0746: /\*\*

0747: \* resources/prefabs/views/MainView.lh

0748: \*/

0749: export class MainViewRTBase extends Laya.Box {

0750: public btnPlay!: Laya.Button;

0751: public btnSkin!: Laya.Button;

0752: public btnTest!: Laya.Button;

0753: public btnHelp!: Laya.Button;

0754: }

0755: const { regClass } = Laya;

0756: import { ConfigPath } from "../../const/ConfigPath";

0757: import { ViewMgr } from "../../core/UI/ViewMgr";

0758: import { Game } from "../Game";

0759: import { EViewKey } from "../ViewConst";

0760: import { ELevelConst } from "../level/LevelConst";

0761: import { MainViewRTBase } from "./MainViewRT.generated";

0762: @regClass()

0763: export class MainViewRT extends MainViewRTBase {

0764: private onClickHelp(): void {

0765: ViewMgr.ins.open(EViewKey.HelpView);

0766: }

0767: private onClickSkin(): void {

0768: ViewMgr.ins.open(EViewKey.SkinView);

0769: }

0770: private onClickPlay(): void {

0771: Game.ins.enterLevel(ELevelConst.Level\_10001);

0772: }

0773: private onClickTest(): void {

0774: Game.ins.enterLevel(ELevelConst.LevelTestId);

0775: }

0776: onAwake(): void {

0777: this.btnHelp.on(Laya.Event.CLICK, this.onClickHelp);

0778: this.btnSkin.on(Laya.Event.CLICK, this.onClickSkin);

0779: this.btnPlay.on(Laya.Event.CLICK, this.onClickPlay);

0780: this.btnTest.on(Laya.Event.CLICK, this.onClickTest);

0781: }

0782: }

0783: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

0784: /\*\*

0785: \* resources/prefabs/views/PauseView.lh

0786: \*/

0787: export class PauseViewRTBase extends Laya.Box {

0788: public lblTitle!: Laya.Label;

0789: public btnClose!: Laya.Button;

0790: public boxPractice!: Laya.Box;

0791: public imgNewRecord!: Laya.Image;

0792: public lblCurrentScore!: Laya.Label;

0793: public lblHistoryScore!: Laya.Label;

0794: public boxNormal!: Laya.Box;

0795: public lblLeftScore!: Laya.Label;

0796: public imgIcon!: Laya.Image;

0797: public btnResume!: Laya.Button;

0798: public btnMainMenu!: Laya.Button;

0799: public btnRestart!: Laya.Button;

0800: }

0801: const { regClass } = Laya;

0802: import { Game } from "../Game";

0803: import { LevelModel } from "../level/LevelModel";

0804: import { SkinModel } from "../skin/SkinModel";

0805: import { PauseViewRTBase } from "./PauseViewRT.generated";

0806: @regClass()

0807: export class PauseViewRT extends PauseViewRTBase {

0808: private updateView(): void {

0809: let isPracticeMode = LevelModel.ins.isPracticeMode();

0810: this.boxNormal.visible = !isPracticeMode;

0811: this.boxPractice.visible = isPracticeMode;

0812: if (isPracticeMode) {

0813: let isNewRecord = LevelModel.ins.checkExistNewRecord();

0814: this.imgNewRecord.visible = isNewRecord;

0815: this.lblCurrentScore.text = LevelModel.ins.currTopDistanceFormat.toString();

0816: isNewRecord && LevelModel.ins.saveNewRecord();

0817: this.lblHistoryScore.text = LevelModel.ins.currHistoryTopDistanceFormat.toString();

0818: } else {

0819: this.imgIcon.skin = SkinModel.ins.getCurrentSkin();

0820: }

0821: }

0822: private onClickResume(): void {

0823: Game.ins.resume();

0824: }

0825: private onClickRestart(): void {

0826: Game.ins.restartLevel();

0827: }

0828: private onClickMainMenu(): void {

0829: Game.ins.backHome();

0830: }

0831: onAwake(): void {

0832: this.btnClose.on(Laya.Event.CLICK, this.onClickResume);

0833: this.btnResume.on(Laya.Event.CLICK, this.onClickResume);

0834: this.btnRestart.on(Laya.Event.CLICK, this.onClickRestart);

0835: this.btnMainMenu.on(Laya.Event.CLICK, this.onClickMainMenu);

0836: }

0837: onEnable(): void {

0838: this.updateView();

0839: }

0840: }

0841: export enum ESkinItemStatus {

0842: Locked,

0843: Idle,

0844: Adventure,

0845: }

0846: export class SkinEvent {

0847: static readonly Unlcok = "Unlcok";

0848: static readonly Adventure = "Adventure";

0849: }

0850: export interface ISkinListData {

0851: id: string;

0852: lblName: string;

0853: imgAvatar: string;

0854: status: ESkinItemStatus;

0855: }

0856: import { Controller } from "../../core/mvc/Controller";

0857: import { SkinModel } from "./SkinModel";

0858: /\*\*

0859: \* author: 陈秀齐

0860: \* time: 2023/12/09 19:32:14

0861: \* desc:

0862: \*/

0863: export class SkinController extends Controller {

0864: private static \_ins: SkinController;

0865: public static get ins(): SkinController {

0866: if (this.\_ins == null) {

0867: this.\_ins = new SkinController();

0868: }

0869: return this.\_ins;

0870: }

0871: private model: SkinModel;

0872: private constructor() {

0873: super();

0874: this.model = SkinModel.ins;

0875: }

0876: unlcok(id: string): void {

0877: this.model.unlock(id);

0878: }

0879: adventure(id: string): void {

0880: this.model.adventure(id);

0881: }

0882: }

0883: const { regClass } = Laya;

0884: import { PathUtils } from "../../utils/PathUtils";

0885: import { ESkinItemStatus } from "./SkinConst";

0886: import { SkinItemRTBase } from "./SkinItemRT.generated";

0887: @regClass()

0888: export class SkinItemRT extends SkinItemRTBase {

0889: get dataSource(): any {

0890: return super.dataSource;

0891: }

0892: set dataSource(value: any) {

0893: super.dataSource = value;

0894: if (!value) return;

0895: let status = value.status;

0896: if (status != null) {

0897: let isIdle = status == ESkinItemStatus.Idle;

0898: let isLocked = status == ESkinItemStatus.Locked;

0899: let isWorking = status == ESkinItemStatus.Adventure;

0900: this.imgBg.skin = PathUtils.getUiImage(isLocked ? "com\_box\_2" : "com\_box\_1");

0901: this.imgAvatarBg.skin = PathUtils.getUiImage(isLocked ? "img\_avatar\_bg\_2" : "img\_avatar\_bg\_1");

0902: this.imgVideo.visible = isLocked;

0903: this.imgAdventuring.visible = isWorking;

0904: this.btnUnlock.visible = isLocked;

0905: this.btnAdventure.visible = isIdle;

0906: }

0907: }

0908: }

0909: /\*\*

0910: \* author: 陈秀齐

0911: \* time: 2023/12/12 10:16:01

0912: \* desc:

0913: \*/

0914: import { LocalData } from "../../utils/LocalData";

0915: export interface ISkinLocalData {

0916: skinId: string;

0917: idleSkinIds: string[];

0918: }

0919: export class SkinLocalData extends LocalData<ISkinLocalData> {

0920: static readonly Key = "SkinLocalData";

0921: static readonly Default: ISkinLocalData = {

0922: skinId: "100001",

0923: idleSkinIds: ["100001"],

0924: }

0925: constructor() {

0926: super(SkinLocalData.Key, SkinLocalData.Default);

0927: }

0928: unlock(id: string): void {

0929: this.data.idleSkinIds.push(id);

0930: this.save();

0931: }

0932: adventure(id: string): void {

0933: this.data.skinId = id;

0934: this.save();

0935: }

0936: }

0937: import { Model } from "../../core/mvc/Model";

0938: import { ConfigUtils, SkinConfigData } from "../../utils/ConfigUtils";

0939: import { PathUtils } from "../../utils/PathUtils";

0940: import { ESkinItemStatus, ISkinListData, SkinEvent } from "./SkinConst";

0941: import { SkinLocalData } from "./SkinLocalData";

0942: /\*\*

0943: \* author: 陈秀齐

0944: \* time: 2023/12/09 19:31:50

0945: \* desc:

0946: \*/

0947: export class SkinModel extends Model {

0948: private static \_ins: SkinModel;

0949: public static get ins(): SkinModel {

0950: if (!this.\_ins) {

0951: this.\_ins = new SkinModel();

0952: }

0953: return this.\_ins;

0954: }

0955: private \_localData: SkinLocalData;

0956: private constructor() {

0957: super();

0958: this.\_localData = new SkinLocalData();

0959: }

0960: checkStatus(id: string): ESkinItemStatus {

0961: let localData = this.\_localData.data;

0962: return localData.skinId === id ? ESkinItemStatus.Adventure : localData.idleSkinIds.some(idleId => idleId === id) ? ESkinItemStatus.Idle : ESkinItemStatus.Locked

0963: }

0964: getList(): ISkinListData[] {

0965: let arr: ISkinListData[] = [];

0966: const configs: SkinConfigData[] = ConfigUtils.get("skin");

0967: for (let i = 0; i < configs.length; i++) {

0968: const conf = configs[i];

0969: arr.push({

0970: id: conf.id,

0971: lblName: conf.name,

0972: status: this.checkStatus(conf.id),

0973: imgAvatar: PathUtils.getAvatar(conf.icon),

0974: });

0975: }

0976: return arr;

0977: }

0978: unlock(id: string): void {

0979: const configs: SkinConfigData[] = ConfigUtils.get("skin");

0980: if (!configs.some(conf => conf.id === id)) {

0981: console.warn("unlock fail!!!, this skin is not exist!", id);

0982: return;

0983: }

0984: this.\_localData.data.idleSkinIds.push(id);

0985: this.\_localData.unlock(id);

0986: this.event(SkinEvent.Unlcok, id);

0987: }

0988: adventure(id: string): void {

0989: const configs: SkinConfigData[] = ConfigUtils.get("skin");

0990: if (!configs.some(conf => conf.id === id)) {

0991: console.warn("unlock fail!!!, this skin is not exist!", id);

0992: return;

0993: }

0994: this.\_localData.adventure(id);

0995: this.event(SkinEvent.Adventure, id);

0996: }

0997: getCurrentSkin(): string {

0998: let id = this.\_localData.data.skinId;

0999: const configs: SkinConfigData[] = ConfigUtils.get("skin");

1000: let conf = configs.find(conf => conf.id === id);

1001: return PathUtils.getAvatar(conf.icon);

1002: }

1003: getCurrentSkinHead(): string {

1004: let id = this.\_localData.data.skinId;

1005: const configs: SkinConfigData[] = ConfigUtils.get("skin");

1006: let conf = configs.find(conf => conf.id === id);

1007: return PathUtils.getHead(conf.icon);

1008: }

1009: }

1010: /\*\*This class is automatically generated by LayaAirIDE, please do not make any modifications. \*/

1011: /\*\*

1012: \* resources/prefabs/views/SkinView.lh

1013: \*/

1014: export class SkinViewRTBase extends Laya.Box {

1015: public lblTitle!: Laya.Label;

1016: public list!: Laya.List;

1017: public btnClose!: Laya.Button;

1018: }

1019: const { regClass } = Laya;

1020: import { ViewMgr } from "../../core/UI/ViewMgr";

1021: import { EViewKey } from "../ViewConst";

1022: import { ESkinItemStatus, ISkinListData, SkinEvent } from "./SkinConst";

1023: import { SkinController } from "./SkinController";

1024: import { SkinModel } from "./SkinModel";

1025: import { SkinViewRTBase } from "./SkinViewRT.generated";

1026: @regClass()

1027: export class SkinViewRT extends SkinViewRTBase {

1028: private onClickClose(): void {

1029: ViewMgr.ins.close(EViewKey.SkinView);

1030: }

1031: private onItemMouse(e: Laya.Event, index: number): void {

1032: if (e.type == Laya.Event.CLICK) {

1033: let listArray = this.list.array as ISkinListData[];

1034: let itemData = listArray[index];

1035: if (e.target.name == "btnAdventure") {

1036: // ctrl 选择皮肤

1037: SkinController.ins.adventure(itemData.id);

1038: } else if (e.target.name == "btnUnlock") {

1039: // ctrl 播放广告解锁皮肤

1040: SkinController.ins.unlcok(itemData.id);

1041: }

1042: }

1043: }

1044: private refreshStatus(id: string): void {

1045: let listArray = this.list.array as ISkinListData[];

1046: let index = listArray.findIndex(data => data.id === id);

1047: let itemData = listArray[index];

1048: itemData.status = SkinModel.ins.checkStatus(id);

1049: this.list.changeItem(index, itemData);

1050: }

1051: private onSkinAdventure(id: string): void {

1052: let listArray = this.list.array as ISkinListData[];

1053: let currAdventure = listArray.find(data => data.status === ESkinItemStatus.Adventure);

1054: this.refreshStatus(currAdventure.id);

1055: this.refreshStatus(id);

1056: }

1057: private onSkinUnlock(id: string): void {

1058: this.refreshStatus(id);

1059: }

1060: onAwake(): void {

1061: this.btnClose.on(Laya.Event.CLICK, this.onClickClose);

1062: this.list.mouseHandler = new Laya.Handler(this, this.onItemMouse, null, false);

1063: }

1064: onEnable(): void {

1065: SkinModel.ins.on(SkinEvent.Unlcok, this, this.onSkinUnlock);

1066: SkinModel.ins.on(SkinEvent.Adventure, this, this.onSkinAdventure);

1067: this.list.array = SkinModel.ins.getList();

1068: }

1069: onDisable(): void {

1070: SkinModel.ins.off(SkinEvent.Unlcok, this, this.onSkinUnlock);

1071: SkinModel.ins.off(SkinEvent.Adventure, this, this.onSkinAdventure);

1072: }

1073: }

1074: /\*\*

1075: \* author: 陈秀齐

1076: \* time: 2023/12/29 10:11:17

1077: \* desc:

1078: \*/

1079: export class ArrayUtils {

1080: static intersection<T>(arrA: T[], arrB: T[]): T[] {

1081: return arrA.filter(i => arrB.indexOf(i) !== -1);

1082: }

1083: static difference<T>(arrA: T[], arrB: T[]): T[] {

1084: return arrA.filter(i => arrB.indexOf(i) === -1);

1085: }

1086: static union<T>(arr1: T[], arr2: T[]): T[] {

1087: return [...new Set([...arr1, ...arr2])];

1088: }

1089: static clear(arr: any[]): void {

1090: arr.length = 0;

1091: }

1092: }

1093: /\*\*

1094: \* author: 陈秀齐

1095: \* time: 2023/12/11 08:57:49

1096: \* desc:

1097: \* TODO:

1098: \* 1.参数key没有被约束；

1099: \* 2.目前是通过json进行加载，需要支持压缩包以及二进制；

1100: \* 3.解析出来目前是any类型的数据；

1101: \*/

1102: import { ConfigPath } from "../const/ConfigPath";

1103: import { TConstructor } from "../core/base/CoreConst";

1104: export class SkinConfigData {

1105: id: string;

1106: name: string;

1107: icon: string;

1108: constructor(data: any) {

1109: this.id = data.id;

1110: this.name = data.name;

1111: this.icon = data.icon;

1112: }

1113: }

1114: export class DialogConfigData {

1115: id: number;

1116: dialogs: { type: number, desc: string }[];

1117: constructor(data: any) {

1118: this.id = data.id;

1119: this.dialogs = data.dialogs;

1120: }

1121: }

1122: export interface IConfigData {

1123: path: string,

1124: cls: TConstructor

1125: }

1126: export const ConfigUtilsMap: { [key: string]: IConfigData } = {

1127: skin: {

1128: path: ConfigPath.JSON\_Skin,

1129: cls: SkinConfigData

1130: },

1131: dialog: {

1132: path: ConfigPath.JSON\_Dialog,

1133: cls: DialogConfigData

1134: },

1135: }

1136: export class ConfigUtils {

1137: private static jsonMap: { [key: string]: any } = {};

1138: static get(key: keyof typeof ConfigUtilsMap) {

1139: if (this.jsonMap[key] == null) {

1140: const { cls, path } = ConfigUtilsMap[key];

1141: let res = Laya.loader.getRes(path);

1142: if (res) {

1143: let arr = [];

1144: for (let key in res.data) {

1145: let data = new cls(res.data[key]);

1146: arr.push(data);

1147: }

1148: this.jsonMap[key] = arr;

1149: }

1150: }

1151: return this.jsonMap[key];

1152: }

1153: }

1154: import { LocalStorageUtils } from "./LocalStorageUtils";

1155: export class LocalData<DATATYPE extends object> {

1156: private \_key: string;

1157: private \_data: DATATYPE;

1158: private \_defaultData: DATATYPE;

1159: constructor(key: string, defaultData: DATATYPE) {

1160: this.\_key = key;

1161: this.\_defaultData = defaultData;

1162: }

1163: get data(): DATATYPE {

1164: if (this.\_data == null) {

1165: this.\_data = LocalStorageUtils.load(this.\_key);

1166: if (this.\_data == null) {

1167: this.\_data = this.\_defaultData;

1168: }

1169: }

1170: return this.\_data;

1171: }

1172: save(): void {

1173: LocalStorageUtils.save(this.\_key, this.data);

1174: }

1175: }

1176: /\*\*

1177: \* author: 陈秀齐

1178: \* time: 2023/12/12 09:31:01

1179: \* desc: 本地数据储存

1180: \* TODO:

1181: \* 1.数据校验，防止篡改和失效；如字段缺失，或违法

1182: \*/

1183: export class LocalStorageUtils {

1184: static \_game: any;

1185: private static GAME\_KEY: string = "\_game\_";

1186: static get game(): any {

1187: if (this.\_game == null) {

1188: this.\_game = Laya.LocalStorage.getJSON(LocalStorageUtils.GAME\_KEY) || {};

1189: }

1190: return this.\_game;

1191: }

1192: private static saveGame(): void {

1193: console.log("saveGame=========", this.game);

1194: Laya.LocalStorage.setJSON(LocalStorageUtils.GAME\_KEY, this.game);

1195: }

1196: static save(key: string, data: any): void {

1197: this.game[key] = data;

1198: Laya.CallLater.I.callLater(this, this.saveGame);

1199: }

1200: static saveNow(key: string, data: any): void {

1201: this.game[key] = data;

1202: Laya.CallLater.I.runCallLater(this, this.saveGame);

1203: }

1204: static load(key: string): any {

1205: return this.game[key];

1206: }

1207: }

1208: /\*\*

1209: \* author: 陈秀齐

1210: \* time: 2023/12/09 19:14:46

1211: \* desc:

1212: \*/

1213: export class MathUtil {

1214: /\*\*

1215: \* int的最大值

1216: \*/

1217: public static INT\_MAX\_VALUE: number = 2147483647;

1218: /\*\*

1219: \* 一弧度的角度数

1220: \*/

1221: public static ONE\_RADIANS: number = 180 / Math.PI;

1222: /\*\*

1223: \* 弧度转换成角度

1224: \* @param radians

1225: \* @return

1226: \*/

1227: public static radiansToDegrees(radians: number): number {

1228: return radians \* this.ONE\_RADIANS;

1229: }

1230: /\*\*

1231: \* 角度转换成弧度

1232: \* @param degrees

1233: \* @return

1234: \*/

1235: public static degreesToRadians(degrees: number): number {

1236: return (degrees \* Math.PI) / 180;

1237: }

1238: /\*\*

1239: \* 得到一个区间的随机数

1240: \* @param min 最小数

1241: \* @param max 最大数

1242: \*/

1243: public static randomF(min: number, max: number): number {

1244: return min + Math.random() \* (max - min);

1245: }

1246: /\*\*

1247: \* 得到一个区间的随机整数,结果包含最小数跟最大数

1248: \* @param min 最小数

1249: \* @param max 最大数

1250: \*/

1251: public static randomI(min: number, max: number): number {

1252: return this.randomF(min, max + 0.99999) >> 0;

1253: }

1254: /\*\*

1255: \* 掷硬币 50%

1256: \* @returns

1257: \*/

1258: public static coinFlip(): boolean {

1259: return this.randomF(0, 1) > 0.5;

1260: }

1261: /\*\*

1262: \* 得到一个数组的随机项

1263: \* @param list

1264: \* @returns

1265: \*/

1266: public static randElement<T>(list: T[]): T | null {

1267: if (list == null || list.length == 0) {

1268: return null;

1269: }

1270: return list[this.randomI(0, list.length - 1)];

1271: }

1272: /\*\*

1273: \* 概率是否发生

1274: \* @param value (0~1)

1275: \* @returns

1276: \*/

1277: public static chance(value: number): boolean {

1278: return Math.random() < value;

1279: }

1280: /\*\*

1281: \* 判断两区间是否部分重叠

1282: \* @param rangeAMin 区间0 起始值

1283: \* @param rangeAMax 区间0 结束值

1284: \* @param rangeBMin 区间1 起始值

1285: \* @param rangeBMax 区间1 起始值

1286: \* @returns

1287: \*/

1288: public static isPartiallyOverlap(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {

1289: return rangeAMin <= rangeBMax && rangeAMax >= rangeBMin;

1290: }

1291: // 判断一个小数区间是否包括另一个小数区间

1292: public static isIntervalIncluding(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {

1293: return rangeAMin <= rangeBMin && rangeAMax >= rangeBMax;

1294: }

1295: // 判断两个小数区间是否不相交也不包括

1296: public static isIntervalsDisjoint(rangeAMin: number, rangeAMax: number, rangeBMin: number, rangeBMax: number): boolean {

1297: return rangeAMax < rangeBMin || rangeAMin > rangeBMax;

1298: }

1299: }

1300: /\*\*

1301: \* author: 陈秀齐

1302: \* time: 2023/12/12 09:40:48

1303: \* desc:

1304: \*/

1305: export class ObjectUtils {

1306: static isEmpty(obj: any): boolean {

1307: return Object.keys(obj).length === 0;

1308: }

1309: }

1310: /\*\*

1311: \* author: 陈秀齐

1312: \* time: 2023/12/20 15:56:33

1313: \* desc:

1314: \*/

1315: export class PathUtils {

1316: static getAvatar(icon: string): string {

1317: return `resources/icon/avatar/${icon}.png`;

1318: }

1319: static getHead(icon: string): string {

1320: return `resources/icon/avatar/${icon}\_head.png`;

1321: }

1322: static getUiImage(icon: string): string {

1323: return `atlas/ui/${icon}.png`;

1324: }

1325: }

1326: /\*\*

1327: \* author: 陈秀齐

1328: \* time: 2023/12/07 19:45:28

1329: \* desc:

1330: \*/

1331: export class StringUtils {

1332: /\*\*

1333: \* 转换成百分比

1334: \* @param decimal 小数

1335: \* @param precision 精度位数 默认两位

1336: \*/

1337: static toPercent(decimal: number, precision: number = 2): string {

1338: return `${(decimal \* 100).toFixed(precision)}%`;

1339: }

1340: static lang(key: string): string {

1341: return Laya.Text.langPacks ? Laya.Text.langPacks[key] : key;

1342: }

1343: static format(template: string, ...values: string[]): string {

1344: return template.replace(/{(\d+)}/g, (match, index) => {

1345: let replacement = values[Number(index)];

1346: replacement = this.lang(replacement);

1347: return typeof replacement !== 'undefined' ? replacement : match;

1348: });

1349: }

1350: }

1351: export class SegmentTree {

1352: root: SegmentTreeNode | null;

1353: constructor() {

1354: this.root = null;

1355: }

1356: // 构建线段树

1357: buildTree(ranges: IRange[]): void {

1358: if (ranges.length === 0) {

1359: return;

1360: }

1361: const min = Math.min(...ranges.map(r => r.min));

1362: const max = Math.max(...ranges.map(r => r.max));

1363: this.root = this.\_buildTree(ranges, min, max);

1364: }

1365: private \_buildTree(ranges: IRange[], min: number, max: number): SegmentTreeNode | null {

1366: if (min > max) {

1367: return null;

1368: }

1369: const root = new SegmentTreeNode(min, max);

1370: if (min < max) {

1371: const mid = Math.floor((min + max) / 2);

1372: root.left = this.\_buildTree(ranges, min, mid);

1373: root.right = this.\_buildTree(ranges, mid + 1, max);

1374: }

1375: // 找到与当前节点区间相交的区间

1376: root.intersects = ranges.filter(

1377: r => r.min <= root.max && r.max >= root.min

1378: );

1379: return root;

1380: }

1381: // 查询与给定区间相交的区间

1382: query(root: SegmentTreeNode | null, min: number, max: number): IRange[] {

1383: const result: IRange[] = [];

1384: if (root === null || min > root.max || max < root.min) {

1385: return result;

1386: }

1387: if (min <= root.min && max >= root.max) {

1388: // 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

1389: return root.intersects;

1390: }

1391: // 否则，递归查询左右子树

1392: result.push(...this.query(root.left, min, max));

1393: result.push(...this.query(root.right, min, max));

1394: return [...new Set(result)];

1395: }

1396: clear(): void {

1397: this.root = null;

1398: }

1399: }

1400: /\*\*

1401: \* author: 陈秀齐

1402: \* time: 2023/12/29 19:09:06

1403: \* desc:

1404: \*/

1405: export interface IRange {

1406: min: number;

1407: max: number;

1408: }

1409: export class SegmentTreeNode {

1410: min: number;

1411: max: number;

1412: left: SegmentTreeNode | null;

1413: right: SegmentTreeNode | null;

1414: intersects: IRange[];

1415: constructor(min: number, max: number) {

1416: this.min = min;

1417: this.max = max;

1418: this.left = null;

1419: this.right = null;

1420: this.intersects = [];

1421: }

1422: }

1423: import { Model } from "./Model";

1424: import { Controller } from "./Controller";

1425: import { TConstructor } from "../base/CoreConst";

1426: export type MOC = Model | Controller;

1427: export class MVCDecorator {

1428: static \_classMap\_: Map<string, TConstructor<MOC>> = new Map();

1429: static \_instanceMap\_: Map<string, MOC> = new Map();

1430: static reg(cls: any): void {

1431: const clsName = (cls as any).name;

1432: if (!(cls.prototype instanceof Controller) && !(cls.prototype instanceof Model)) {

1433: console.warn("mvc reg warning!!! register a class not model or ctrl", clsName, cls);

1434: return;

1435: }

1436: if (MVCDecorator.\_classMap\_.has(clsName)) {

1437: console.warn("mvc reg warning!!! repeat register", clsName, cls);

1438: return;

1439: }

1440: MVCDecorator.\_classMap\_.set(clsName, cls);

1441: }

1442: static prop(cls: Controller | Model): PropertyDecorator {

1443: return (target: any, key: PropertyKey) => {

1444: const getter = function () {

1445: const clsName = (cls as any).name;

1446: if (!MVCDecorator.\_classMap\_.has(clsName)) {

1447: console.warn("mvc prop warning!!! try to get a model no register", clsName, key, cls);

1448: return;

1449: }

1450: const cacheClass = MVCDecorator.\_classMap\_.get(clsName);

1451: let instance = MVCDecorator.\_instanceMap\_.get(clsName);

1452: if (!MVCDecorator.\_instanceMap\_.has(clsName)) {

1453: instance = new cacheClass();

1454: MVCDecorator.\_instanceMap\_.set(clsName, instance);

1455: }

1456: return instance;

1457: };

1458:

1459: const setter = function () {

1460: const clsName = (cls as any).name;

1461: console.warn("mvc prop warning!!! try to set a prop.", clsName, key, cls);

1462: };

1463: // Redefine the property with the new getter and setter

1464: Object.defineProperty(target, key, {

1465: get: getter,

1466: set: setter,

1467: enumerable: true,

1468: configurable: true,

1469: });

1470: }

1471: }

1472: // todo

1473: private static model\_handler(cls: Model, event: string) {

1474: return (target: Object, propertyKey: string | symbol, descriptor: any) => {

1475: console.log("model\_handler========", target, propertyKey, descriptor);

1476: }

1477: }

1478: }

1479: import { NodeFlags } from "../Const"

1480: import { Component } from "../components/Component"

1481: import { Event } from "../events/Event"

1482: import { EventDispatcher } from "../events/EventDispatcher"

1483: import { Pool } from "../utils/Pool"

1484: import { Stat } from "../utils/Stat"

1485: import { Timer } from "../utils/Timer"

1486: import { ILaya } from "../../ILaya";

1487: import { ComponentDriver } from "../components/ComponentDriver";

1488: const ARRAY\_EMPTY: any[] = [];

1489: export type Callback = ((...args: any[]) => Promise<void>) | ((...args: any[]) => void) | undefined;

1490: export interface ITransition<STATE, EVENT, CALLBACK> {

1491: fromState: STATE;

1492: event: EVENT;

1493: toState: STATE;

1494: cb: CALLBACK;

1495: }

1496: export function t<STATE, EVENT, CALLBACK>(

1497: fromState: STATE, event: EVENT, toState: STATE,

1498: cb?: CALLBACK): ITransition<STATE, EVENT, CALLBACK> {

1499: return { fromState, event, toState, cb };

1500: }

1501: /\*\*

1502: \* <p> <code>Pool</code> 是对象池类，用于对象的存储、重复使用。</p>

1503: \* <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>

1504: \*/

1505: export class Pool {

1506: /\*\*@private \*/

1507: private static \_CLSID: number = 0;

1508: /\*\*@private \*/

1509: private static POOLSIGN: string = "\_\_InPool";

1510: /\*\*@private 对象存放池。\*/

1511: private static \_poolDic: any = {};

1512: /\*\*

1513: \* 根据对象类型标识字符，获取对象池。

1514: \* @param sign 对象类型标识字符。

1515: \* @return 对象池。

1516: \*/

1517: static getPoolBySign(sign: string): any[] {

1518: return Pool.\_poolDic[sign] || (Pool.\_poolDic[sign] = []);

1519: }

1520: /\*\*

1521: \* 清除对象池的对象。

1522: \* @param sign 对象类型标识字符。

1523: \*/

1524: static clearBySign(sign: string): void {

1525: if (Pool.\_poolDic[sign]) Pool.\_poolDic[sign].length = 0;

1526: }

1527: /\*\*

1528: \* 将对象放到对应类型标识的对象池中。

1529: \* @param sign 对象类型标识字符。

1530: \* @param item 对象。

1531: \*/

1532: static recover(sign: string, item: any): void {

1533: 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

1534: return;

1535: item[Pool.POOLSIGN] = true;

1536: Pool.getPoolBySign(sign).push(item);

1537: }

1538: /\*\*

1539: \* 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

1540: \* @param instance 类的具体实例

1541: \*/

1542: static recoverByClass(instance: any): void {

1543: if (instance) {

1544: var className: string = instance["\_\_className"] || instance.constructor.\_$gid;

1545: if (className) Pool.recover(className, instance);

1546: }

1547: }

1548: /\*\*

1549: \* 返回类的唯一标识

1550: \*/

1551: private static \_getClassSign(cla: any): string {

1552: var className = cla["\_\_className"] || cla["\_$gid"];

1553: if (!className) {

1554: cla["\_$gid"] = className = Pool.\_CLSID + "";

1555: Pool.\_CLSID++;

1556: }

1557: return className;

1558: }

1559: /\*\*

1560: \* 根据类型创建对象

1561: \* @param cls 类型

1562: \*/

1563: static createByClass<T>(cls: new () => T): T {

1564: return Pool.getItemByClass(Pool.\_getClassSign(cls), cls);

1565: }

1566: /\*\*

1567: \* <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>

1568: \* <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>

1569: \* @param sign 对象类型标识字符。

1570: \* @param cls 用于创建该类型对象的类。

1571: \* @return 此类型标识的一个对象。

1572: \*/

1573: static getItemByClass<T>(sign: string, cls: new () => T): T {

1574: let rst: any;

1575: let pool = Pool.getPoolBySign(sign);

1576: if (pool.length)

1577: rst = pool.pop();

1578: else

1579: rst = new cls();

1580: rst[Pool.POOLSIGN] = false;

1581: return rst;

1582: }

1583: /\*\*

1584: \* <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>

1585: \* <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>

1586: \* @param sign 对象类型标识字符。

1587: \* @param createFun 用于创建该类型对象的方法。

1588: \* @param caller this对象

1589: \* @return 此类型标识的一个对象。

1590: \*/

1591: static getItemByCreateFun(sign: string, createFun: Function, caller: any = null): any {

1592: var pool: any[] = Pool.getPoolBySign(sign);

1593: var rst: any = pool.length ? pool.pop() : createFun.call(caller);

1594: rst[Pool.POOLSIGN] = false;

1595: return rst;

1596: }

1597: /\*\*

1598: \* 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 。

1599: \* @param sign 对象类型标识字符。

1600: \* @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 。

1601: \*/

1602: static getItem(sign: string): any {

1603: var pool: any[] = Pool.getPoolBySign(sign);

1604: var rst: any = pool.length ? pool.pop() : null;

1605: if (rst) {

1606: rst[Pool.POOLSIGN] = false;

1607: }

1608: return rst;

1609: }

1610: }

1611: /\*\*

1612: \* 添加到父对象后调度。

1613: \* @eventType Event.ADDED

1614: \*/

1615: /\*[Event(name = "added", type = "laya.events.Event")]\*/

1616: /\*\*

1617: \* 被父对象移除后调度。

1618: \* @eventType Event.REMOVED

1619: \*/

1620: /\*[Event(name = "removed", type = "laya.events.Event")]\*/

1621: /\*\*

1622: \* 加入节点树时调度。

1623: \* @eventType Event.DISPLAY

1624: \*/

1625: /\*[Event(name = "display", type = "laya.events.Event")]\*/

1626: /\*\*

1627: \* 从节点树移除时调度。

1628: \* @eventType Event.UNDISPLAY

1629: \*/

1630: /\*[Event(name = "undisplay", type = "laya.events.Event")]\*/

1631: /\*\*

1632: \* <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。

1633: \*/

1634: export class Node extends EventDispatcher {

1635: static EVENT\_SET\_ACTIVESCENE: string = "ActiveScene";

1636: static EVENT\_SET\_IN\_ACTIVESCENE: string = "InActiveScene";

1637: /\*\*@private \*/

1638: private \_bits: number = 0;

1639: /\*\*@private \*/

1640: private \_hideFlags: number = 0;

1641: /\*\*@internal 子对象集合，请不要直接修改此对象。\*/

1642: \_children: Node[] = ARRAY\_EMPTY;

1643: /\*\*@internal 父节点对象\*/

1644: \_parent: Node = null;

1645: /\*\*@internal \*/

1646: \_destroyed: boolean = false;

1647: /\*\*@internal \*/

1648: \_conchData: any;

1649: /\*\*@internal \*/

1650: \_componentDriver: ComponentDriver;

1651: /\*\*@internal \*/

1652: \_is3D: boolean;

1653: \_url: string;

1654: \_extra: INodeExtra;

1655: /\*\*节点名称。\*/

1656: name: string = "";

1657: /\*\* 节点标签 \*/

1658: tag: string;

1659: /\*\*

1660: \* 如果节点从资源中创建，这里记录是他的url

1661: \*/

1662: get url(): string {

1663: return this.\_url;

1664: }

1665: /\*\*

1666: \* 设置资源的URL

1667: \*/

1668: set url(path: string) {

1669: this.\_url = path;

1670: }

1671: get hideFlags(): number {

1672: return this.\_hideFlags;

1673: }

1674: set hideFlags(value: number) {

1675: this.\_hideFlags = value;

1676: }

1677: /\*\* 是否3D节点，即Scene3D和Sprite3D及其衍生类 \*/

1678: get is3D(): boolean {

1679: return this.\_is3D;

1680: }

1681: /\*\* 是否已经销毁。对象销毁后不能再使用。\*/

1682: get destroyed(): boolean {

1683: return this.\_destroyed;

1684: }

1685: constructor() {

1686: super();

1687: this.\_initialize();

1688: }

1689: //@internal

1690: \_initialize(): void {

1691: this.\_extra = {};

1692: }

1693: \_setBit(type: number, value: boolean): void {

1694: if (type === NodeFlags.DISPLAY) {

1695: var preValue: boolean = this.\_getBit(type);

1696: if (preValue != value) this.\_updateDisplayedInstage();

1697: }

1698: if (value) this.\_bits |= type;

1699: else this.\_bits &= ~type;

1700: }

1701: \_getBit(type: number): boolean {

1702: return (this.\_bits & type) != 0;

1703: }

1704: /\*\*@internal \*/

1705: \_setUpNoticeChain(): void {

1706: if (this.\_getBit(NodeFlags.DISPLAY)) this.\_setBitUp(NodeFlags.DISPLAY);

1707: }

1708: /\*\*@internal \*/

1709: \_setBitUp(type: number): void {

1710: var ele: Node = this;

1711: ele.\_setBit(type, true);

1712: ele = ele.\_parent;

1713: while (ele) {

1714: if (ele.\_getBit(type)) return;

1715: ele.\_setBit(type, true);

1716: ele = ele.\_parent;

1717: }

1718: }

1719: protected onStartListeningToType(type: string) {

1720: if (type === Event.DISPLAY || type === Event.UNDISPLAY) {

1721: if (!this.\_getBit(NodeFlags.DISPLAY)) this.\_setBitUp(NodeFlags.DISPLAY);

1722: }

1723: }

1724: bubbleEvent(type: string, data?: any) {

1725: let arr: Array<Node> = \_bubbleChainPool.length > 0 ? \_bubbleChainPool.pop() : [];

1726: arr.length = 0;

1727: let obj: Node = this;

1728: while (obj) {

1729: if (obj.activeInHierarchy)

1730: arr.push(obj);

1731: obj = obj.parent;

1732: }

1733: if (data instanceof Event) {

1734: data.\_stopped = false;

1735: for (let obj of arr) {

1736: data.setTo(type, obj, this);

1737: obj.event(type, data);

1738: if (data.\_stopped)

1739: break;

1740: }

1741: }

1742: else {

1743: for (let obj of arr)

1744: obj.event(type, data);

1745: }

1746: \_bubbleChainPool.push(arr);

1747: }

1748: hasHideFlag(flag: number): boolean {

1749: return (this.\_hideFlags & flag) != 0;

1750: }

1751: /\*\*

1752: \* <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>

1753: \* <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>

1754: \* @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。

1755: \*/

1756: destroy(destroyChild: boolean = true): void {

1757: this.\_destroyed = true;

1758: this.destroyAllComponent();

1759: this.\_parent && this.\_parent.removeChild(this);

1760: //销毁子节点

1761: if (this.\_children) {

1762: if (destroyChild) this.destroyChildren();

1763: else this.removeChildren();

1764: }

1765: this.onDestroy();

1766: this.\_children = null;

1767: //移除所有事件监听

1768: this.offAll();

1769: }

1770: /\*\*

1771: \* 销毁时执行

1772: \* 此方法为虚方法，使用时重写覆盖即可

1773: \*/

1774: onDestroy(): void {

1775: //trace("onDestroy node", this.name);

1776: }

1777: /\*\*

1778: \* 销毁所有子对象，不销毁自己本身。

1779: \*/

1780: destroyChildren(): void {

1781: //销毁子节点

1782: if (this.\_children) {

1783: //为了保持销毁顺序，所以需要正序销毁

1784: for (let i = 0, n = this.\_children.length; i < n; i++) {

1785: this.\_children[0] && this.\_children[0].destroy(true);

1786: }

1787: }

1788: }

1789: /\*\*

1790: \* 添加子节点。

1791: \* @param node 节点对象

1792: \* @return 返回添加的节点

1793: \*/

1794: addChild<T extends Node>(node: T): T {

1795: if (!node || this.\_destroyed || node as any === this) return node;

1796: if ((<any>node).\_zOrder) this.\_setBit(NodeFlags.HAS\_ZORDER, true);

1797: if (node.\_parent === this) {

1798: var index: number = this.getChildIndex(node);

1799: if (index !== this.\_children.length - 1) {

1800: this.\_children.splice(index, 1);

1801: this.\_children.push(node);

1802: this.\_childChanged();

1803: }

1804: } else {

1805: node.\_parent && node.\_parent.removeChild(node);

1806: this.\_children === ARRAY\_EMPTY && (this.\_children = []);

1807: this.\_children.push(node);

1808: node.\_setParent(this);

1809: }

1810: return node;

1811: }

1812: /\*\*

1813: \* 批量增加子节点

1814: \* @param ...args 无数子节点。

1815: \*/

1816: addChildren(...args: any[]): void {

1817: var i: number = 0, n: number = args.length;

1818: while (i < n) {

1819: this.addChild(args[i++]);

1820: }

1821: }

1822: /\*\*

1823: \* 添加子节点到指定的索引位置。

1824: \* @param node 节点对象。

1825: \* @param index 索引位置。

1826: \* @return 返回添加的节点。

1827: \*/

1828: addChildAt(node: Node, index: number): Node {

1829: if (!node || this.\_destroyed || node === this) return node;

1830: if ((<any>node).\_zOrder) this.\_setBit(NodeFlags.HAS\_ZORDER, true);

1831: if (index >= 0 && index <= this.\_children.length) {

1832: if (node.\_parent === this) {

1833: var oldIndex: number = this.getChildIndex(node);

1834: this.\_children.splice(oldIndex, 1);

1835: this.\_children.splice(index, 0, node);

1836: this.\_childChanged();

1837: } else {

1838: node.\_parent && node.\_parent.removeChild(node);

1839: this.\_children === ARRAY\_EMPTY && (this.\_children = []);

1840: this.\_children.splice(index, 0, node);

1841: node.\_setParent(this);

1842: }

1843: return node;

1844: } else {

1845: throw new Error("appendChildAt:The index is out of bounds");

1846: }

1847: }

1848: /\*\*

1849: \* 根据子节点对象，获取子节点的索引位置。

1850: \* @param node 子节点。

1851: \* @return 子节点所在的索引位置。

1852: \*/

1853: getChildIndex(node: Node): number {

1854: return this.\_children.indexOf(node);

1855: }

1856: /\*\*

1857: \* 根据子节点的名字，获取子节点对象。

1858: \* @param name 子节点的名字。

1859: \* @return 节点对象。

1860: \*/

1861: getChildByName(name: string): Node {

1862: for (let child of this.\_children) {

1863: if (child && child.name === name)

1864: return child;

1865: }

1866: return null;

1867: }

1868: /\*\*

1869: \* 根据子节点的索引位置，获取子节点对象。

1870: \* @param index 索引位置

1871: \* @return 子节点

1872: \*/

1873: getChildAt(index: number): Node {

1874: return this.\_children[index] || null;

1875: }

1876: /\*\*

1877: \* 设置子节点的索引位置。

1878: \* @param node 子节点。

1879: \* @param index 新的索引。

1880: \* @return 返回子节点本身。

1881: \*/

1882: setChildIndex(node: Node, index: number): Node {

1883: var childs: any[] = this.\_children;

1884: if (index < 0 || index >= childs.length) {

1885: throw new Error("setChildIndex:The index is out of bounds.");

1886: }

1887: var oldIndex: number = this.getChildIndex(node);

1888: if (oldIndex < 0) throw new Error("setChildIndex:node is must child of this object.");

1889: childs.splice(oldIndex, 1);

1890: childs.splice(index, 0, node);

1891: this.\_childChanged();

1892: return node;

1893: }

1894: /\*\*

1895: \* 子节点发生改变。

1896: \* @private

1897: \* @param child 子节点。

1898: \*/

1899: protected \_childChanged(child: Node = null): void {

1900: }

1901: /\*\*

1902: \* 删除子节点。

1903: \* @param node 子节点

1904: \* @return 被删除的节点

1905: \*/

1906: removeChild(node: Node): Node {

1907: if (!this.\_children) return node;

1908: var index: number = this.\_children.indexOf(node);

1909: return this.removeChildAt(index);

1910: }

1911: /\*\*

1912: \* 从父容器删除自己，如已经被删除不会抛出异常。

1913: \* @return 当前节点（ Node ）对象。

1914: \*/

1915: removeSelf(): Node {

1916: this.\_parent && this.\_parent.removeChild(this);

1917: return this;

1918: }

1919: /\*\*

1920: \* 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。

1921: \* @param name 对象名字。

1922: \* @return 查找到的节点（ Node ）对象。

1923: \*/

1924: removeChildByName(name: string): Node {

1925: var node: Node = this.getChildByName(name);

1926: node && this.removeChild(node);

1927: return node;

1928: }

1929: /\*\*

1930: \* 根据子节点索引位置，删除对应的子节点对象。

1931: \* @param index 节点索引位置。

1932: \* @return 被删除的节点。

1933: \*/

1934: removeChildAt(index: number): Node {

1935: var node: Node = this.getChildAt(index);

1936: if (node) {

1937: this.\_children.splice(index, 1);

1938: node.\_setParent(null);

1939: }

1940: return node;

1941: }

1942: /\*\*

1943: \* 删除指定索引区间的所有子对象。

1944: \* @param beginIndex 开始索引。

1945: \* @param endIndex 结束索引。

1946: \* @return 当前节点对象。

1947: \*/

1948: removeChildren(beginIndex: number = 0, endIndex: number = 0x7fffffff): Node {

1949: if (this.\_children && this.\_children.length > 0) {

1950: var childs: any[] = this.\_children;

1951: if (beginIndex === 0 && endIndex >= childs.length - 1) {

1952: var arr: any[] = childs;

1953: this.\_children = ARRAY\_EMPTY;

1954: } else {

1955: arr = childs.splice(beginIndex, endIndex - beginIndex + 1);

1956: }

1957: for (var i: number = 0, n: number = arr.length; i < n; i++) {

1958: arr[i].\_setParent(null);

1959: }

1960: }

1961: return this;

1962: }

1963: /\*\*

1964: \* 替换子节点。

1965: \* 将传入的新节点对象替换到已有子节点索引位置处。

1966: \* @param newNode 新节点。

1967: \* @param oldNode 老节点。

1968: \* @return 返回新节点。

1969: \*/

1970: replaceChild(newNode: Node, oldNode: Node): Node {

1971: var index: number = this.\_children.indexOf(oldNode);

1972: if (index > -1) {

1973: this.\_children.splice(index, 1, newNode);

1974: oldNode.\_setParent(null);

1975: newNode.\_setParent(this);

1976: return newNode;

1977: }

1978: return null;

1979: }

1980: /\*\*

1981: \* 子对象数量。

1982: \*/

1983: get numChildren(): number {

1984: return this.\_children ? this.\_children.length : 0;

1985: }

1986: /\*\*父节点。\*/

1987: get parent(): Node {

1988: return this.\_parent;

1989: }

1990: /\*\*检查本节点是否是某个节点的上层节点

1991: \* @param node

1992: \* @return

1993: \*/

1994: isAncestorOf(node: Node): boolean {

1995: let p = node.parent;

1996: while (p) {

1997: if (p == this)

1998: return true;

1999: p = p.parent;

2000: }

2001: return false;

2002: };

2003: /\*\*@private \*/

2004: protected \_setParent(value: Node): void {

2005: if (this.\_parent !== value) {

2006: if (value) {

2007: this.\_parent = value;

2008: //如果父对象可见，则设置子对象可见

2009: this.\_onAdded();

2010: this.event(Event.ADDED);

2011: if (this.\_getBit(NodeFlags.DISPLAY)) {

2012: this.\_setUpNoticeChain();

2013: value.displayedInStage && this.\_displayChild(this, true);

2014: }

2015: value.\_childChanged(this);

2016: } else {

2017: //设置子对象不可见

2018: this.\_onRemoved();

2019: this.event(Event.REMOVED);

2020: let p = this.\_parent;

2021: if (this.\_getBit(NodeFlags.DISPLAY)) this.\_displayChild(this, false);

2022: this.\_parent = value;

2023: p.\_childChanged(this);

2024: }

2025: }

2026: }

2027: /\*\*表示是否在显示列表中显示。\*/

2028: get displayedInStage(): boolean {

2029: if (this.\_getBit(NodeFlags.DISPLAY)) return this.\_getBit(NodeFlags.DISPLAYED\_INSTAGE);

2030: this.\_setBitUp(NodeFlags.DISPLAY);

2031: return this.\_getBit(NodeFlags.DISPLAYED\_INSTAGE);

2032: }

2033: /\*\*@private \*/

2034: private \_updateDisplayedInstage(): void {

2035: var ele: Node;

2036: ele = this;

2037: var stage: Node = ILaya.stage;

2038: var displayedInStage: boolean = false;

2039: while (ele) {

2040: if (ele.\_getBit(NodeFlags.DISPLAY)) {

2041: displayedInStage = ele.\_getBit(NodeFlags.DISPLAYED\_INSTAGE);

2042: break;

2043: }

2044: if (ele === stage || ele.\_getBit(NodeFlags.DISPLAYED\_INSTAGE)) {

2045: displayedInStage = true;

2046: break;

2047: }

2048: ele = ele.\_parent;

2049: }

2050: this.\_setBit(NodeFlags.DISPLAYED\_INSTAGE, displayedInStage);

2051: }

2052: /\*\*@internal \*/

2053: \_setDisplay(value: boolean): void {

2054: if (this.\_getBit(NodeFlags.DISPLAYED\_INSTAGE) !== value) {

2055: this.\_setBit(NodeFlags.DISPLAYED\_INSTAGE, value);

2056: if (value) this.event(Event.DISPLAY);

2057: else this.event(Event.UNDISPLAY);

2058: }

2059: }

2060: /\*\*

2061: \* 设置指定节点对象是否可见(是否在渲染列表中)。

2062: \* @private

2063: \* @param node 节点。

2064: \* @param display 是否可见。

2065: \*/

2066: private \_displayChild(node: Node, display: boolean): void {

2067: var childs: any[] = node.\_children;

2068: if (childs) {

2069: for (var i: number = 0, n: number = childs.length; i < n; i++) {

2070: var child: Node = childs[i];

2071: if (!child) continue;

2072: if (!child.\_getBit(NodeFlags.DISPLAY)) continue;

2073: if (child.\_children.length > 0) {

2074: this.\_displayChild(child, display);

2075: } else {

2076: child.\_setDisplay(display);

2077: }

2078: }

2079: }

2080: node.\_setDisplay(display);

2081: }

2082: /\*\*

2083: \* 当前容器是否包含指定的 <code>Node</code> 节点对象 。

2084: \* @param node 指定的 <code>Node</code> 节点对象 。

2085: \* @return 一个布尔值表示是否包含指定的 <code>Node</code> 节点对象 。

2086: \*/

2087: contains(node: Node): boolean {

2088: if (node === this) return true;

2089: while (node) {

2090: if (node.\_parent === this) return true;

2091: node = node.\_parent;

2092: }

2093: return false;

2094: }

2095: /\*\*

2096: \* 定时重复执行某函数。功能同Laya.timer.timerLoop()。

2097: \* @param delay 间隔时间(单位毫秒)。

2098: \* @param caller 执行域(this)。

2099: \* @param method 结束时的回调方法。

2100: \* @param args （可选）回调参数。

2101: \* @param coverBefore （可选）是否覆盖之前的延迟执行，默认为true。

2102: \* @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

2103: \*/

2104: timerLoop(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true, jumpFrame: boolean = false): void {

2105: this.timer.loop(delay, caller, method, args, coverBefore, jumpFrame);

2106: }

2107: /\*\*

2108: \* 定时执行某函数一次。功能同Laya.timer.timerOnce()。

2109: \* @param delay 延迟时间(单位毫秒)。

2110: \* @param caller 执行域(this)。

2111: \* @param method 结束时的回调方法。

2112: \* @param args （可选）回调参数。

2113: \* @param coverBefore （可选）是否覆盖之前的延迟执行，默认为true。

2114: \*/

2115: timerOnce(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {

2116: this.timer.\_create(false, false, delay, caller, method, args, coverBefore);

2117: }

2118: /\*\*

2119: \* 定时重复执行某函数(基于帧率)。功能同Laya.timer.frameLoop()。

2120: \* @param delay 间隔几帧(单位为帧)。

2121: \* @param caller 执行域(this)。

2122: \* @param method 结束时的回调方法。

2123: \* @param args （可选）回调参数。

2124: \* @param coverBefore （可选）是否覆盖之前的延迟执行，默认为true。

2125: \*/

2126: frameLoop(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {

2127: this.timer.\_create(true, true, delay, caller, method, args, coverBefore);

2128: }

2129: /\*\*

2130: \* 定时执行一次某函数(基于帧率)。功能同Laya.timer.frameOnce()。

2131: \* @param delay 延迟几帧(单位为帧)。

2132: \* @param caller 执行域(this)

2133: \* @param method 结束时的回调方法

2134: \* @param args （可选）回调参数

2135: \* @param coverBefore （可选）是否覆盖之前的延迟执行，默认为true

2136: \*/

2137: frameOnce(delay: number, caller: any, method: Function, args: any[] = null, coverBefore: boolean = true): void {

2138: this.timer.\_create(true, false, delay, caller, method, args, coverBefore);

2139: }

2140: /\*\*

2141: \* 清理定时器。功能同Laya.timer.clearTimer()。

2142: \* @param caller 执行域(this)。

2143: \* @param method 结束时的回调方法。

2144: \*/

2145: clearTimer(caller: any, method: Function): void {

2146: this.timer.clear(caller, method);

2147: }

2148: /\*\*

2149: \* <p>延迟运行指定的函数。</p>

2150: \* <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>

2151: \* @param method 要执行的函数的名称。例如，functionName。

2152: \* @param args 传递给 <code>method</code> 函数的可选参数列表。

2153: \*

2154: \* @see #runCallLater()

2155: \*/

2156: callLater(method: Function, args: any[] = null): void {

2157: this.timer.callLater(this, method, args);

2158: }

2159: /\*\*

2160: \* <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>

2161: \* @param method 要执行的函数名称。例如，functionName。

2162: \* @see #callLater()

2163: \*/

2164: runCallLater(method: Function): void {

2165: this.timer.runCallLater(this, method);

2166: }

2167: //============================组件化支持==============================

2168: /\*\* @private \*/

2169: protected \_components: Component[];

2170: /\*\*@private \*/

2171: private \_activeChangeScripts: Component[];

2172: \_scene: Node;

2173: /\*\*

2174: \* 获得所属场景。

2175: \* @return 场景。

2176: \*/

2177: get scene(): any {

2178: return this.\_scene;

2179: }

2180: /\*\*

2181: \* 获取自身是否激活。

2182: \* @return 自身是否激活。

2183: \*/

2184: get active(): boolean {

2185: return !this.\_getBit(NodeFlags.NOT\_READY) && !this.\_getBit(NodeFlags.NOT\_ACTIVE);

2186: }

2187: /\*\*

2188: \* 设置是否激活。

2189: \* @param value 是否激活。

2190: \*/

2191: set active(value: boolean) {

2192: value = !!value;

2193: if (!this.\_getBit(NodeFlags.NOT\_ACTIVE) !== value) {

2194: if (this.\_activeChangeScripts && this.\_activeChangeScripts.length !== 0) {

2195: if (value)

2196: 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.";

2197: else

2198: 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.";

2199: } else {

2200: this.\_setBit(NodeFlags.NOT\_ACTIVE, !value);

2201: if (this.\_parent) {

2202: if (this.\_parent.activeInHierarchy) {

2203: this.\_processActive(value, true);

2204: }

2205: }

2206: }

2207: }

2208: }

2209: /\*\*

2210: \* 获取在场景中是否激活。

2211: \* @return 在场景中是否激活。

2212: \*/

2213: get activeInHierarchy(): boolean {

2214: return this.\_getBit(NodeFlags.ACTIVE\_INHIERARCHY);

2215: }

2216: /\*\*

2217: \* @private

2218: \*/

2219: protected \_onActive(): void {

2220: Stat.spriteCount++;

2221: }

2222: /\*\*

2223: \* @private

2224: \*/

2225: protected \_onInActive(): void {

2226: Stat.spriteCount--;

2227: }

2228: /\*\*

2229: \* @private

2230: \*/

2231: protected \_onActiveInScene(): void {

2232: this.event(Node.EVENT\_SET\_ACTIVESCENE, this.\_scene);

2233: //override it.

2234: }

2235: /\*\*

2236: \* @private

2237: \*/

2238: protected \_onInActiveInScene(): void {

2239: this.event(Node.EVENT\_SET\_IN\_ACTIVESCENE, this.\_scene);

2240: //override it.

2241: }

2242: /\*\*

2243: \* 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

2244: \* 此方法为虚方法，使用时重写覆盖即可

2245: \*/

2246: onAwake(): void {

2247: //this.name && trace("onAwake node ", this.name);

2248: }

2249: /\*\*

2250: \* 组件被启用后执行，比如节点被添加到舞台后

2251: \* 此方法为虚方法，使用时重写覆盖即可

2252: \*/

2253: onEnable(): void {

2254: //this.name && trace("onEnable node ", this.name);

2255: }

2256: /\*\*

2257: \* 组件被禁用时执行，比如从节点从舞台移除后

2258: \* 此方法为虚方法，使用时重写覆盖即可

2259: \*/

2260: onDisable(): void {

2261: //trace("onDisable node", this.name);

2262: }

2263: /\*\*

2264: \* @internal

2265: \*/

2266: \_parse(data: any, spriteMap: any): void {

2267: //override it.

2268: }

2269: /\*\*

2270: \* @internal

2271: \*/

2272: \_setBelongScene(scene: Node): void {

2273: if (!this.\_scene || this.scene != scene) {

2274: this.\_scene = scene;

2275: this.\_onActiveInScene();

2276: for (let i = 0, n = this.\_children.length; i < n; i++)

2277: this.\_children[i].\_setBelongScene(scene);

2278: }

2279: }

2280: /\*\*

2281: \* @internal

2282: \*/

2283: \_setUnBelongScene(): void {

2284: if (this.\_scene !== this) {//移除节点本身是scene不继续派发

2285: this.\_onInActiveInScene();

2286: this.\_scene = null;

2287: for (let i = 0, n = this.\_children.length; i < n; i++)

2288: this.\_children[i].\_setUnBelongScene();

2289: }

2290: }

2291: \_processActive(active: boolean, fromSetter?: boolean) {

2292: (this.\_activeChangeScripts) || (this.\_activeChangeScripts = []);

2293: let arr = this.\_activeChangeScripts;

2294: if (active)

2295: this.\_activeHierarchy(arr, fromSetter);

2296: else

2297: this.\_inActiveHierarchy(arr, fromSetter);

2298: for (let i = 0, n = arr.length; i < n; i++) {

2299: let comp = arr[i];

2300: comp.owner && comp.\_setActive(active);

2301: }

2302: arr.length = 0;

2303: }

2304: /\*\*

2305: \* @internal

2306: \*/

2307: \_activeHierarchy(activeChangeScripts: any[], fromSetter?: boolean): void {

2308: this.\_setBit(NodeFlags.ACTIVE\_INHIERARCHY, true);

2309: if (this.\_components) {

2310: for (let i = 0, n = this.\_components.length; i < n; i++) {

2311: let comp = this.\_components[i];

2312: if (comp.\_isScript())

2313: (comp.\_enabled) && (activeChangeScripts.push(comp));

2314: else

2315: comp.\_setActive(true);

2316: }

2317: }

2318: this.\_onActive();

2319: for (let i = 0, n = this.\_children.length; i < n; i++) {

2320: let child = this.\_children[i];

2321: (!child.\_getBit(NodeFlags.NOT\_ACTIVE) && !child.\_getBit(NodeFlags.NOT\_READY)) && (child.\_activeHierarchy(activeChangeScripts, fromSetter));

2322: }

2323: if (!this.\_getBit(NodeFlags.AWAKED)) {

2324: this.\_setBit(NodeFlags.AWAKED, true);

2325: this.onAwake();

2326: }

2327: this.onEnable();

2328: }

2329: /\*\*

2330: \* @internal

2331: \*/

2332: \_inActiveHierarchy(activeChangeScripts: any[], fromSetter?: boolean): void {

2333: this.\_onInActive();

2334: if (this.\_components) {

2335: for (let i = 0, n = this.\_components.length; i < n; i++) {

2336: let comp = this.\_components[i];

2337: if (comp.\_isScript())

2338: comp.\_enabled && (activeChangeScripts.push(comp));

2339: else

2340: comp.\_setActive(false);

2341: }

2342: }

2343: this.\_setBit(NodeFlags.ACTIVE\_INHIERARCHY, false);

2344: for (let i = 0, n = this.\_children.length; i < n; i++) {

2345: let child = this.\_children[i];

2346: (child && !child.\_getBit(NodeFlags.NOT\_ACTIVE)) && (child.\_inActiveHierarchy(activeChangeScripts, fromSetter));

2347: }

2348: this.onDisable();

2349: }

2350: /\*\*

2351: \* @private

2352: \*/

2353: protected \_onAdded(): void {

2354: if (this.\_activeChangeScripts && this.\_activeChangeScripts.length !== 0) {

2355: 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.";

2356: } else {

2357: let parentScene = this.\_parent.scene;

2358: parentScene && this.\_setBelongScene(parentScene);

2359: (this.\_parent.activeInHierarchy && this.active) && this.\_processActive(true);

2360: }

2361: }

2362: /\*\*

2363: \* @private

2364: \*/

2365: protected \_onRemoved(): void {

2366: if (this.\_activeChangeScripts && this.\_activeChangeScripts.length !== 0) {

2367: 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.";

2368: } else {

2369: (this.\_parent.activeInHierarchy && this.active) && this.\_processActive(false);

2370: this.\_parent.scene && this.\_setUnBelongScene();

2371: }

2372: }

2373: /\*\*

2374: \* @internal

2375: \*/

2376: \_addComponentInstance(comp: Component): void {

2377: if (!this.\_components)

2378: this.\_components = [];

2379: this.\_components.push(comp);

2380: comp.\_setOwner(this);

2381: if (this.activeInHierarchy)

2382: comp.\_setActive(true);

2383: this.\_componentsChanged?.(comp, 0);

2384: }

2385: /\*\*

2386: \* @internal

2387: \*/

2388: \_destroyComponent(comp: Component) {

2389: if (!this.\_components)

2390: return;

2391: let i = this.\_components.indexOf(comp);

2392: if (i != -1) {

2393: this.\_components.splice(i, 1);

2394: comp.\_destroy();

2395: this.\_componentsChanged?.(comp, 1);

2396: }

2397: }

2398: /\*\*

2399: \* @internal

2400: \*/

2401: private destroyAllComponent(): void {

2402: if (!this.\_components)

2403: return;

2404: for (let i = 0, n = this.\_components.length; i < n; i++) {

2405: let item = this.\_components[i];

2406: item && !item.destroyed && item.\_destroy();

2407: }

2408: this.\_components.length = 0;

2409: this.\_componentsChanged?.(null, 2);

2410: }

2411: /\*\*

2412: \* 组件列表发生改变。

2413: \* @private

2414: \*/

2415: protected \_componentsChanged?(comp: Component, action: 0 | 1 | 2): void;

2416: /\*\*

2417: \* @internal 克隆。

2418: \* @param destObject 克隆源。

2419: \*/

2420: \_cloneTo(destObject: any, srcRoot: Node, dstRoot: Node): void {

2421: var destNode: Node = (<Node>destObject);

2422: if (this.\_components) {

2423: for (let i = 0, n = this.\_components.length; i < n; i++) {

2424: var destComponent = destNode.addComponent((this.\_components[i] as any).constructor);

2425: this.\_components[i].\_cloneTo(destComponent);

2426: }

2427: }

2428: }

2429: /\*\*

2430: \* 添加组件实例。

2431: \* @param component 组建实例。

2432: \* @return 组件。

2433: \*/

2434: addComponentInstance(component: Component): Component {

2435: if (component.owner)

2436: throw "Node:the component has belong to other node.";

2437: if (component.\_singleton && this.getComponent(((<any>component)).constructor))

2438: console.warn("Node:the component is singleton, can't add the second one.", component);

2439: else

2440: this.\_addComponentInstance(component);

2441: return component;

2442: }

2443: /\*\*

2444: \* 添加组件。

2445: \* @param componentType 组件类型。

2446: \* @return 组件。

2447: \*/

2448: addComponent<T extends Component>(componentType: new () => T): T {

2449: let comp: T = Pool.createByClass(componentType);

2450: if (!comp) {

2451: throw "missing " + componentType.toString();

2452: }

2453: if (comp.\_singleton && this.getComponent(componentType))

2454: console.warn("Node:the component is singleton, can't add the second one.", comp);

2455: else

2456: this.\_addComponentInstance(comp);

2457: return comp;

2458: }

2459: /\*\*

2460: \* 获得组件实例，如果没有则返回为null

2461: \* @param componentType 组建类型

2462: \* @return 返回组件

2463: \*/

2464: getComponent<T extends Component>(componentType: new () => T): T {

2465: if (this.\_components) {

2466: for (let i = 0, n = this.\_components.length; i < n; i++) {

2467: let comp = this.\_components[i];

2468: if (comp instanceof componentType)

2469: return comp;

2470: }

2471: }

2472: return null;

2473: }

2474: /\*\*

2475: \* 返回所有组件实例。

2476: \* @return 返回组件实例数组。

2477: \*/

2478: get components(): ReadonlyArray<Component> {

2479: return this.\_components || ARRAY\_EMPTY;

2480: }

2481: /\*\*

2482: \* 获得组件实例，如果没有则返回为null

2483: \* @param componentType 组件类型

2484: \* @return 返回组件数组

2485: \*/

2486: getComponents(componentType: typeof Component): Component[] {

2487: var arr: any[];

2488: if (this.\_components) {

2489: for (let i = 0, n = this.\_components.length; i < n; i++) {

2490: let comp = this.\_components[i];

2491: if (comp instanceof componentType) {

2492: arr = arr || [];

2493: arr.push(comp);

2494: }

2495: }

2496: }

2497: return arr;

2498: }

2499: /\*\*

2500: \* 获取timer

2501: \*/

2502: get timer(): Timer {

2503: return this.\_scene ? this.\_scene.timer : ILaya.timer;

2504: }

2505: /\*\*

2506: \* 反序列化后会调用

2507: \*/

2508: onAfterDeserialize() { }

2509: }

2510: const \_bubbleChainPool: Array<Array<Node>> = [];

2511: export interface INodeExtra { }

2512: /\*\*

2513: \* <p><code>Handler</code> 是事件处理器类。</p>

2514: \* <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>

2515: \* <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>

2516: \*/

2517: export class Handler {

2518: /\*\*@private handler对象池\*/

2519: protected static \_pool: Handler[] = [];

2520: /\*\*@private \*/

2521: private static \_gid: number = 1;

2522: /\*\* 执行域(this)。\*/

2523: caller: Object | null;

2524: /\*\* 处理方法。\*/

2525: method: Function | null;

2526: /\*\* 参数。\*/

2527: args: any[] | null;

2528: /\*\* 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 。\*/

2529: once = false;

2530: /\*\*@private \*/

2531: protected \_id = 0;

2532: /\*\*

2533: \* 根据指定的属性值，创建一个 <code>Handler</code> 类的实例。

2534: \* @param caller 执行域。

2535: \* @param method 处理函数。

2536: \* @param args 函数参数。

2537: \* @param once 是否只执行一次。

2538: \*/

2539: constructor(caller: Object | null = null, method: Function | null = null, args: any[] | null = null, once: boolean = false) {

2540: this.setTo(caller, method, args, once);

2541: }

2542: /\*\*

2543: \* 设置此对象的指定属性值。

2544: \* @param caller 执行域(this)。

2545: \* @param method 回调方法。

2546: \* @param args 携带的参数。

2547: \* @param once 是否只执行一次，如果为true，执行后执行recover()进行回收。

2548: \* @return 返回 handler 本身。

2549: \*/

2550: setTo(caller: any, method: Function | null, args: any[] | null, once = false): Handler {

2551: this.\_id = Handler.\_gid++;

2552: this.caller = caller;

2553: this.method = method;

2554: this.args = args;

2555: this.once = once;

2556: return this;

2557: }

2558: /\*\*

2559: \* 执行处理器。

2560: \*/

2561: run(): any {

2562: if (this.method == null) return null;

2563: var id: number = this.\_id;

2564: var result: any = this.method.apply(this.caller, this.args);

2565: this.\_id === id && this.once && this.recover();

2566: return result;

2567: }

2568: /\*\*

2569: \* 执行处理器，并携带额外数据。

2570: \* @param data 附加的回调数据，可以是单数据或者Array(作为多参)。

2571: \*/

2572: runWith(data: any): any {

2573: if (this.method == null) return null;

2574: var id: number = this.\_id;

2575: if (data == null)

2576: var result: any = this.method.apply(this.caller, this.args);

2577: else if (!this.args && !data.unshift) result = this.method.call(this.caller, data);

2578: else if (this.args) result = this.method.apply(this.caller, this.args.concat(data));

2579: else result = this.method.apply(this.caller, data);

2580: this.\_id === id && this.once && this.recover();

2581: return result;

2582: }

2583: /\*\*

2584: \* 清理对象引用。

2585: \*/

2586: clear(): Handler {

2587: this.caller = null;

2588: this.method = null;

2589: this.args = null;

2590: return this;

2591: }

2592: /\*\*

2593: \* 清理并回收到 Handler 对象池内。

2594: \*/

2595: recover(): void {

2596: if (this.\_id > 0) {

2597: this.\_id = 0;

2598: Handler.\_pool.push(this.clear());

2599: }

2600: }

2601: /\*\*

2602: \* 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。

2603: \* @param caller 执行域(this)。

2604: \* @param method 回调方法。

2605: \* @param args 携带的参数。

2606: \* @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。

2607: \* @return 返回创建的handler实例。

2608: \*/

2609: static create(caller: any, method: Function | null, args: any[] | null = null, once: boolean = true): Handler {

2610: if (Handler.\_pool.length)

2611: return (Handler.\_pool.pop() as Handler).setTo(caller, method, args, once);

2612: return new Handler(caller, method, args, once);

2613: }

2614: }

2615: import { ConfigPath } from "../const/ConfigPath";

2616: import { ViewMgr } from "../core/UI/ViewMgr";

2617: import { Game } from "../views/Game";

2618: import { EViewKey } from "../views/ViewConst";

2619: import { ELevelConst } from "../views/level/LevelConst";

2620: import { LevelModel } from "../views/level/LevelModel";

2621: import { BackgroundRoot } from "./BackgroundRoot";

2622: import { InputManager } from "./InputManager";

2623: import { EItemType } from "./Item/EItemType";

2624: import { LevelCamera } from "./LevelCamera";

2625: import { Player } from "./Player";

2626: import { LevelNodeManager } from "./levelParse/LevelNodeManager";

2627: /\*\*

2628: \* author: 陈秀齐

2629: \* time: 2023/12/14 11:05:26

2630: \* desc:

2631: \*/

2632: const { regClass, property } = Laya;

2633: @regClass('81d36ae9-c41d-47cc-b112-cc4568ccd384', '../src/level/Level.ts')

2634: export class Level extends Laya.Script {

2635: declare owner: Laya.Sprite;

2636: @property({ type: Laya.Sprite, tips: "地面根节点" })

2637: private moveRoot: Laya.Sprite;

2638: @property({ type: Laya.Sprite, tips: "地面根节点" })

2639: private groundRoot: Laya.Sprite;

2640: @property({ type: Laya.Sprite, tips: "障碍物根节点" })

2641: private obstacleRoot: Laya.Sprite;

2642: @property({ type: Laya.Sprite, tips: "物品根节点" })

2643: private itemRoot: Laya.Sprite;

2644: @property({ type: Laya.Sprite, tips: "特效根节点" })

2645: private effectRoot: Laya.Sprite;

2646: @property({ type: Laya.Sprite, tips: "ui根节点" })

2647: private uiRoot: Laya.Sprite;

2648: @property({ type: InputManager, tips: "关卡输入控制" })

2649: private inputManager: InputManager;

2650: @property({ type: LevelNodeManager, tips: "关卡节点管理" })

2651: private nodeManager: LevelNodeManager;

2652: @property({ type: Player, tips: "角色" })

2653: private player: Player;

2654: @property({ type: LevelCamera, tips: "相机控制" })

2655: public levelCamera: LevelCamera;

2656: @property({ type: ["Record", Number], tips: "配置" })

2657: public config: Record<string, number>;

2658: @property({ type: Laya.Animation, tips: "受击特效" })

2659: private animHurt: Laya.Animation;

2660: @property({ type: Laya.Animation, tips: "落地特效" })

2661: private animDust: Laya.Animation;

2662: public backgroundRoot: BackgroundRoot;

2663: private \_isInit: boolean = false;

2664: private \_enabledCollision: boolean = true;

2665: get spawnPoint(): [number, number] {

2666: const { x, y } = this.player.owner;

2667: return [x, y];

2668: }

2669: private parsePrefabData(levelId: ELevelConst, offset: number): void {

2670: this.nodeManager.init(levelId, offset);

2671: }

2672: private checkCollision(): void {

2673: const playerRect = this.player.collisionBox;

2674: //

2675: let items = this.nodeManager.items;

2676: let item = this.tryCheckCollision(playerRect, items);

2677: // trigger

2678: if (item && item.type == EItemType.FinalAward) {

2679: Game.ins.win();

2680: this.\_enabledCollision = false;

2681: this.player.stop();

2682: this.player.pause();

2683: };

2684: if (item && item.type == EItemType.FoCat) {

2685: item.collisionEvent();

2686: };

2687: // collision

2688: let obstacles = this.nodeManager.obstacles;

2689: let obstacle = this.tryCheckCollision(playerRect, obstacles);

2690: if (obstacle) {

2691: this.inputManager.cancel();

2692: this.player.addForce(obstacle.force, obstacle.degrees);

2693: this.showHurtEffect();

2694: Laya.SoundManager.playSound(ConfigPath.M\_CatHurt);

2695: return;

2696: };

2697: let grounds = this.nodeManager.grounds;

2698: let scrollGrounds = grounds.filter(g => g.isScrollBack);

2699: let staticGrounds = grounds.filter(g => !g.isScrollBack);

2700: let ground = this.tryCheckCollision(playerRect, scrollGrounds);

2701: if (!ground) {

2702: ground = this.tryCheckCollision(playerRect, staticGrounds);

2703: }

2704: const newIsGround = ground != null;

2705: const lastIsGround = this.player.isGround;

2706: if (lastIsGround != newIsGround) {

2707: this.player.isGround = newIsGround;

2708: // 落地

2709: if (newIsGround) {

2710: if (ground.isScrollBack) {

2711: this.player.velocityY = 0;

2712: this.player.velocityX = -ground.moveSpeed;

2713: } else {

2714: this.player.stop();

2715: }

2716: this.recordPlayerPos();

2717: this.showDustEffect();

2718: this.player.owner.y = ground.owner.y;

2719: Laya.SoundManager.playSound(ConfigPath.M\_Foot);

2720: }

2721: }

2722: }

2723: private tryCheckCollision<T extends { collisionBox: Laya.Rectangle }>(playerRect: Laya.Rectangle, list: T[]): T {

2724: return list.find(i => i.collisionBox.intersects(playerRect));

2725: }

2726: private onAnimDustComplete(): void {

2727: this.animDust.visible = false;

2728: }

2729: private onAnimHurtComplete(): void {

2730: this.animHurt.visible = false;

2731: }

2732: onAwake(): void {

2733: this.animDust.on(Laya.Event.COMPLETE, this, this.onAnimDustComplete);

2734: this.animHurt.on(Laya.Event.COMPLETE, this, this.onAnimHurtComplete);

2735: }

2736: onUpdate(): void {

2737: if (!this.\_isInit) return;

2738: if (this.\_enabledCollision) {

2739: this.checkCollision();

2740: }

2741: }

2742: onDestroy(): void {

2743: LevelModel.ins.resetDistance();

2744: }

2745: init(levelId: number, backgroundRoot: BackgroundRoot): void {

2746: if (this.\_isInit) return;

2747: this.\_isInit = true;

2748: // levelId = ELevelConst.Level\_10002;

2749: LevelModel.ins.currId = levelId;

2750: let startLine = this.config[levelId];

2751: this.moveRoot.x = startLine;

2752: this.uiRoot.x = -startLine;

2753: let realStartPos = startLine - this.spawnPoint[0];

2754: LevelModel.ins.setStartSpace(realStartPos);

2755: this.parsePrefabData(levelId, startLine);

2756: this.player.spawn(...this.spawnPoint);

2757: this.inputManager.init(this.player);

2758: this.levelCamera.init(this.player);

2759: this.levelCamera.addFollower(backgroundRoot);

2760: this.backgroundRoot = backgroundRoot;

2761: }

2762: recordPlayerPos(): void {

2763: LevelModel.ins.recordPlayerPos(this.levelCamera.distance);

2764: }

2765: reEnterLevel(levelId: number): void {

2766: LevelModel.ins.currId = levelId;

2767: this.levelCamera.backToStart();

2768: LevelModel.ins.resetDistance();

2769: let startLine = this.config[levelId];

2770: this.moveRoot.x = startLine;

2771: this.uiRoot.x = -startLine;

2772: let realStartPos = startLine - this.spawnPoint[0];

2773: LevelModel.ins.setStartSpace(realStartPos);

2774: this.nodeManager.clear();

2775: this.parsePrefabData(levelId, startLine);

2776: this.\_enabledCollision = true;

2777: this.player.resume();

2778: }

2779: restart(): void {

2780: this.reEnterLevel(LevelModel.ins.currId);

2781: }

2782: scrollTo(pos: number): void {

2783: this.player.hide();

2784: this.\_enabledCollision = false;

2785: this.inputManager.enabled = false;

2786: LevelModel.ins.isScrollClose = true;

2787: ViewMgr.ins.close(EViewKey.HudView);

2788: this.levelCamera.scrollTo(pos, Laya.Handler.create(this, () => {

2789: this.player.show();

2790: this.\_enabledCollision = true;

2791: this.inputManager.enabled = true;

2792: LevelModel.ins.scrollEnd();

2793: ViewMgr.ins.open(EViewKey.HudView);

2794: LevelModel.ins.isScrollClose = false;

2795: }, null, true));

2796: }

2797: showDustEffect(): void {

2798: this.animDust.visible = true;

2799: let point = this.player.getFootPoint(this.effectRoot);

2800: this.animDust.pos(point.x, point.y);

2801: this.animDust.play(0, false);

2802: }

2803: showHurtEffect(): void {

2804: this.animHurt.visible = true;

2805: let point = this.player.getFootPoint(this.effectRoot);

2806: this.animHurt.pos(point.x, point.y);

2807: this.animHurt.play(0, false);

2808: }

2809: }

2810: /\*\*

2811: \* author: 陈秀齐

2812: \* time: 2023/12/12 15:22:31

2813: \* desc: 背景视差移动

2814: \* MTC

2815: \* todo:

2816: \* 1.屏幕宽度适应问题

2817: \*/

2818: const { regClass, property } = Laya;

2819: @regClass()

2820: export class Background extends Laya.Script {

2821: declare owner: Laya.Image;

2822: @property({ type: Number, tips: "移动视差比例" })

2823: moveScale: number = 1;

2824: @property({ type: Number, tips: "纹理默认宽度" })

2825: textureWidth: number = 720;

2826: @property({ type: Number, tips: "初始宽度为原始宽度的倍数" })

2827: repeatX: number = 3;

2828: private \_startPosX: number = 0;

2829: private get distance(): number {

2830: return Math.abs(this.owner.x - this.\_startPosX);

2831: }

2832: private resetPos(): void {

2833: let real = this.owner.x - this.\_startPosX;

2834: this.owner.x = this.\_startPosX + real % this.textureWidth;

2835: }

2836: private isOutOfBounds(): boolean {

2837: return this.distance \* this.repeatX > this.textureWidth;

2838: }

2839: onStart(): void {

2840: const stageW = this.textureWidth \* this.repeatX;

2841: this.owner.x = -stageW;

2842: this.owner.width = stageW \* this.repeatX;

2843: this.\_startPosX = this.owner.x;

2844: }

2845: move(distance: number): void {

2846: this.owner.x += distance \* this.moveScale;

2847: if (this.isOutOfBounds()) {

2848: this.resetPos();

2849: }

2850: }

2851: }

2852: import { Background } from "./Background";

2853: import { CameraFollower } from "./CameraFollower";

2854: /\*\*

2855: \* author: 陈秀齐

2856: \* time: 2023/12/12 21:00:37

2857: \* desc:

2858: \*/

2859: const { regClass, property } = Laya;

2860: @regClass()

2861: export class BackgroundRoot extends CameraFollower {

2862: declare owner: Laya.Sprite;

2863: @property({ type: [Background], tips: "视差背景图层集合" })

2864: backgrounds: Background[] = [];

2865: @property({ type: Boolean, tips: "是否自动移动" })

2866: autoMove: boolean = false;

2867: @property({ type: Number, tips: "自动移动的速度" })

2868: autoMoveSpeed: number = 0;

2869: move(distance: number): void {

2870: if (distance == 0) return;

2871: this.backgrounds.forEach(bg => bg.move(distance));

2872: }

2873: onUpdate(): void {

2874: if (this.autoMove) {

2875: this.move(this.autoMoveSpeed);

2876: }

2877: }

2878: randomSkin(): void {

2879: this.setSkin(Math.floor(Math.random() \* 3));

2880: }

2881: setSkin(index: number): void {

2882: this.backgrounds.forEach((bg, i) => {

2883: bg.owner.skin = `resources/scene/bg${index}/Layer\_${i}.png`;

2884: });

2885: }

2886: enterAnim(): void {

2887: Laya.Tween.from(this.owner, { alpha: 0 }, 1000)

2888: }

2889: exitAnim(): void {

2890: Laya.Tween.from(this.owner, { alpha: 0 }, 1000)

2891: }

2892: }

2893: import { CameraFollower } from "./CameraFollower";

2894: /\*\*

2895: \* author: 陈秀齐

2896: \* time: 2023/12/13 08:48:23

2897: \* desc:

2898: \*/

2899: export interface ICameraFocusTarget {

2900: velocityX: number;

2901: }