NZA 开发者指南 V1.0

NoZeroAI Role Asset Packaging Specification Developer Guide

版本：V1.0

发布日期：2026 年 5 月 1 日

生效日期：2026 年 6 月 1 日

发布机构：NZA 标准委员会（零越无界 / NoZeroAI）

官方网站：www.nza.tech

重要声明：本指南是对《NZA 文件格式规范 V1.0》第 9 章的实施扩展与示例补充。本指南中的任何内容均不构成标准的强制约束。若本指南中的示例代码与主规范定义冲突，以主规范为准。

目录

1 快速入门

1.1 什么是 NZA

NZA（NoZeroAI Role Asset Packaging Specification）是一套面向 AI 角色资产的统一表达与封装标准。它解决了角色设定在创建、编辑、发布、分发与多平台适配过程中常见的结构不统一、版本混乱、兼容性差等问题。

NZA 的核心是六层 Canonical Role Schema，它将角色资产的语义信息进行了结构化分层，确保角色的身份、性格、记忆、行为策略和安全边界能够被清晰地表达和传递。

1.2 第一个 NZA 包的创建

创建一个简单的 NZA 包只需要以下几个步骤：

1. 定义角色的六层 Schema 数据

2. 填写包清单信息
3. 填写兼容性信息
4. 计算完整性摘要
5. 保存为.nza.json 文件

创建一个简单的 NZA 包（Python）

```Python
import json
import hashlib
from datetime import datetime

def create_simple_nza_package(role_name, role_summary,
identity_setting):
    """创建一个简单的 NZA 包"""

    # 1. 定义六层 Schema 数据
    schema_data = {
        "layer1_identity": {
            "name": role_name,
            "summary": role_summary
        },
        "layer2_role": {
            "identitySetting": identity_setting
        },
        "layer3_personality": {
            "toneAndStyle": ["友好", "专业"]
        },
        "layer4_memory": {
            "staticMemories": []
        },
        "layer5_behaviorStrategy": {
            "scenarioStrategies": []
        },
        "layer6_constraints": {
            "safetyPolicy": ["不得生成违法违规内容", "不得泄露个人敏感
信息"]
        }
    }

    # 2. 填写包清单信息
    now = datetime.utcnow().isoformat() + "Z"
    manifest = {
        "specVersion": "1.0",
```

```python
        "schemaVersion": "1.0",
        "packageId": f"role_pkg_{hashlib.md5((role_name +
now).encode()).hexdigest()[:8]}",
        "packageType": "role_asset",
        "status": "draft",
        "versionNo": "v1.0.0",
        "createdAt": now,
        "updatedAt": now
    }

    # 3. 填写资产元信息
    asset_meta = {
        "name": role_name,
        "description": role_summary
    }

    # 4. 填写兼容性信息
    compatibility = {
        "minReaderSpecVersion": "1.0",
        "degradationPolicy": {
            "onMissingMemorySupport": "warn",
            "onMissingToolGuardrails": "warn",
            "onConstraintLoss": "block",
            "onStyleLoss": "warn"
        }
    }

    # 5. 构建 NZA 包
    nza_package = {
        "nza_manifest": manifest,
        "nza_payload": {
            "assetMeta": asset_meta,
            "schemaData": schema_data
        },
        "nza_compatibility": compatibility,
        "nza_integrity": {}  # 先留空，计算摘要时会填充
    }

    # 6. 计算完整性摘要
    nza_package = calculate_integrity(nza_package)

    return nza_package

def calculate_integrity(nza_package):
    """计算 NZA 包的完整性摘要"""
```

```
    # 暂时移除 digest 和 signature 字段
    temp_package = nza_package.copy()
    if "digest" in temp_package["nza_integrity"]:
        del temp_package["nza_integrity"]["digest"]
    if "signature" in temp_package["nza_integrity"]:
        del temp_package["nza_integrity"]["signature"]

    # 执行 RFC 8785 规范化序列化
    canonical_json = json.dumps(temp_package, ensure_ascii=False,
sort_keys=True, separators=(',', ':'))

    # 计算 SHA-256 摘要
    digest =
hashlib.sha256(canonical_json.encode('utf-8')).hexdigest()

    # 填充完整性信息
    nza_package["nza_integrity"] = {
        "algorithm": "SHA-256",
        "canonicalization": "RFC 8785",
        "scope": "package_without_integrity_digest",
        "digest": digest
    }

    return nza_package

# 使用示例
if __name__ == "__main__":
    package = create_simple_nza_package(
        "小明",
        "一个友好的客服助手",
        "专业的在线客服，负责解答用户的常见问题"
    )

    with open("xiaoming.nza.json", "w", encoding="utf-8") as f:
        json.dump(package, f, ensure_ascii=False, indent=2)

    print("NZA 包创建成功：xiaoming.nza.json")
```

1.3 常见平台适配器使用

NZA 标准提供了多种平台的适配器，可以将标准 NZA 包转换为对应平台可识别的格式。

使用 OpenClaw 适配器导出 NZA 包（Python）

```python
import json
from nza.adapters import OpenClawAdapter

def export_to_openclaw(nza_package_path, output_path):
    """将 NZA 包导出为 OpenClaw 平台格式"""

    # 读取 NZA 包
    with open(nza_package_path, "r", encoding="utf-8") as f:
        nza_package = json.load(f)

    # 创建适配器
    adapter = OpenClawAdapter()

    # 执行兼容性校验
    validation_result = adapter.validate(nza_package)
    if validation_result["errors"]:
        print("兼容性校验失败：")
        for error in validation_result["errors"]:
            print(f"  - {error}")
        return False

    if validation_result["warnings"]:
        print("兼容性校验警告：")
        for warning in validation_result["warnings"]:
            print(f"  - {warning}")

    # 执行导出
    openclaw_format = adapter.export(nza_package)

    # 保存结果
    with open(output_path, "w", encoding="utf-8") as f:
        json.dump(openclaw_format, f, ensure_ascii=False, indent=2)

    print(f"导出成功：{output_path}")
    return True

# 使用示例
if __name__ == "__main__":
    export_to_openclaw("xiaoming.nza.json",
"xiaoming_openclaw.json")
```

2 传统 Prompt 与 NZA 六层 Schema 映射指南

2.1 映射原则

1. **非替代，而是结构化**：NZA 不消灭自然语言 Prompt，而是将隐含逻辑显性化、可治理、可校验
2. **混编友好**：Schema 字段内可直接使用自然语言文本，兼容现有写作习惯
3. **边界清晰**：身份、性格、记忆、策略、约束严格分层，避免语义混叠
4. **渐进式迁移**：支持从"纯文本 Prompt"到"半结构化"再到"全结构化"的逐步迁移

2.2 常见场景映射对照表

传统 Prompt 写法（自然语言）	NZA 对应层级	建议结构化写法（Schema）	核心优势
你叫小明，是一个 20 岁的大学生，性格开朗	Layer1 身份	```json	
"layer1_identity": {			
"name": "小明",			
"summary": "20 岁开朗大学生"			
}			

```
Plain Text
|如果用户问敏感问题，请拒绝回答，并说"我不方便回答这个问题"|Layer6 约束
|```json
"layer6_constraints": {
  "safetyPolicy": [
    "禁止回答政治敏感话题",
    "拒绝回答格式：我不方便回答这个问题"
  ]
}
```|安全边界最高优先级，不可绕过|
|用户喜欢蓝色，讨厌吃香菜|Layer4 静态记忆|```json
"layer4_memory": {
  "staticMemories": [
    "用户偏好：蓝色",
```
```

````
    "用户禁忌：香菜"
  ]
}
```|记忆可版本化、可迁移、不污染运行时|
|如果用户生气了，先道歉，然后提供三个解决方案|Layer5 行为策略|```json
"layer5_behaviorStrategy": {
  "scenarioStrategies": [{
    "scenario": "用户生气",
    "goal": "安抚用户情绪并解决问题",
    "approach": ["道歉", "提供方案 A/B/C"]
  }]
}
```|逻辑流程化，跨平台稳定执行|
|说话要幽默风趣，多用网络流行语|Layer3 性格偏好|```json
"layer3_personality": {
  "toneAndStyle": ["幽默风趣", "网络流行语"]
}
```|语气可量化、可适配、可降级|
|你是我的私人健身教练，负责制定训练计划和饮食建议|Layer2 角色定位
|```json
"layer2_role": {
  "identitySetting": "私人健身教练",
  "relationshipSetting": "用户的健康顾问"
}
```|场景与关系明确，不越界|

### 2.3 自动映射工具使用说明

NZA 标准委员会提供开源的 Prompt→NZA 自动映射工具，可将纯文本 Prompt 自动转
换为初步的六层 Schema 结构。工具输出结果需经过人工审阅与修正，确保语义准确
性。

工具地址：工具将随标准发布后在 NZA 官方 GitHub 组织下提供，请访问
www.nza.tech/tools 获取最新信息。

**使用自动映射工具转换 Prompt（Python）**
```python
from nza.tools import PromptToNZAMapper

def prompt_to_nza(prompt_text):
    """将纯文本 Prompt 转换为 NZA 包"""

    # 创建映射器
````
````

```
    mapper = PromptToNZAMapper()

    # 执行映射
    result = mapper.map(prompt_text)

    print("映射结果：")
    print(f"总体置信度：
{result['analysisMeta']['overallConfidence']}%")

    if result['analysisMeta']['needsConfirmation']:
        print("\n 需要人工确认的项：")
        for item in result['analysisMeta']['needsConfirmation']:
            print(f"  - {item}")

    if result['analysisMeta']['conflicts']:
        print("\n 检测到的冲突：")
        for conflict in result['analysisMeta']['conflicts']:
            print(f"  - {conflict}")

    return result['nza_package']

# 使用示例
if __name__ == "__main__":
    prompt = """
    你叫小红，是一个温柔的心理咨询师。
    你擅长倾听用户的烦恼，给予温暖的安慰和专业的建议。
    说话要温柔、耐心，多用鼓励的语气。
    如果用户有自杀倾向，请立即引导他们寻求专业帮助。
    不要代替用户做决定，而是帮助他们自己找到解决方案。
    """

    nza_package = prompt_to_nza(prompt)

    with open("xiaohong.nza.json", "w", encoding="utf-8") as f:
        json.dump(nza_package, f, ensure_ascii=False, indent=2)

    print("\nNZA 包已保存：xiaohong.nza.json")
```

2.4 手动优化技巧

1. **分层检查**：确保每个字段都放在了正确的层级，避免语义混层
2. **必填字段优先**：先填写所有必填字段，再填写可选字段
3. **场景具体化**：在 Layer5 中尽可能具体地描述各种场景的处理策略

4. **安全边界明确**：在 Layer6 中明确列出所有安全约束和禁止事项
5. **版本控制**：每次修改都更新版本号和更新时间
6. **测试验证**：在目标平台上测试导出后的角色行为是否符合预期

2.5 代码示例：Prompt 转 NZA 转换器

简单的 Prompt 转 NZA 转换器实现（Python）

```python
import re
import hashlib
import json
from datetime import datetime

class SimplePromptToNZAMapper:
    """简单的 Prompt 转 NZA 转换器"""

    def __init__(self):
        # 定义各层的关键词模式
        self.patterns = {
            "name": r"你叫([^，。！？]+)",
            "identity": r"你是([^，。！？]+)",
            "tone": r"说话要([^，。！？]+)",
            "safety": r"如果用户[^，。！？]*请([^。！？]+)",
            "behavior": r"不要([^，。！？]+)"
        }

    def map(self, prompt_text):
        """将 Prompt 转换为 NZA 包"""

        # 提取信息
        name_match = re.search(self.patterns["name"], prompt_text)
        identity_match = re.search(self.patterns["identity"],
prompt_text)
        tone_matches = re.findall(self.patterns["tone"],
prompt_text)
        safety_matches = re.findall(self.patterns["safety"],
prompt_text)
        behavior_matches = re.findall(self.patterns["behavior"],
prompt_text)

        # 构建 Schema 数据
        schema_data = {
```

```
            "layer1_identity": {
                "name": name_match.group(1) if name_match else "未命
名角色",
                "summary": identity_match.group(1) if identity_match
else "通用 AI 助手"
            },
            "layer2_role": {
                "identitySetting": identity_match.group(1) if
identity_match else "通用 AI 助手"
            },
            "layer3_personality": {
                "toneAndStyle": tone_matches if tone_matches else ["
友好", "专业"]
            },
            "layer4_memory": {
                "staticMemories": []
            },
            "layer5_behaviorStrategy": {
                "scenarioStrategies": [],
                "refusalStrategy": "遇到无法回答的问题时，礼貌地说明并提
供相关帮助。"
            },
            "layer6_constraints": {
                "safetyPolicy": ["不得生成违法违规内容", "不得泄露个人
敏感信息"] + safety_matches
            }
        }

        # 构建分析元数据
        analysis_meta = {
            "overallConfidence": 70,
            "layerConfidence": {
                "layer1_identity": 80 if name_match else 50,
                "layer2_role": 80 if identity_match else 50,
                "layer3_personality": 75,
                "layer4_memory": 0,
                "layer5_behaviorStrategy": 60,
                "layer6_constraints": 85
            },
            "needsConfirmation": [],
            "conflicts": [],
            "sourceSummary": {
                "sourceType": "prompt_import"
            }
```

```
        }

        # 构建完整的 NZA 包
        now = datetime.utcnow().isoformat() + "Z"
        package_id = 
f"role_pkg_{hashlib.md5((schema_data['layer1_identity']['name'] + 
now).encode()).hexdigest()[:8]}"

        nza_package = {
            "nza_manifest": {
                "specVersion": "1.0",
                "schemaVersion": "1.0",
                "packageId": package_id,
                "packageType": "role_asset",
                "status": "draft",
                "versionNo": "v1.0.0",
                "createdAt": now,
                "updatedAt": now
            },
            "nza_payload": {
                "assetMeta": {
                    "name": schema_data["layer1_identity"]["name"],
                    "description": 
schema_data["layer1_identity"]["summary"]
                },
                "schemaData": schema_data,
                "analysisMeta": analysis_meta
            },
            "nza_compatibility": {
                "minReaderSpecVersion": "1.0",
                "degradationPolicy": {
                    "onMissingMemorySupport": "warn",
                    "onMissingToolGuardrails": "warn",
                    "onConstraintLoss": "block",
                    "onStyleLoss": "warn"
                }
            },
            "nza_integrity": {}
        }

        # 计算完整性摘要
        nza_package = self._calculate_integrity(nza_package)

        return {
            "nza_package": nza_package,
```

```
            "analysisMeta": analysis_meta
        }

    def _calculate_integrity(self, nza_package):
        """计算完整性摘要"""
        temp_package = nza_package.copy()
        if "digest" in temp_package["nza_integrity"]:
            del temp_package["nza_integrity"]["digest"]

        canonical_json = json.dumps(temp_package,
ensure_ascii=False, sort_keys=True, separators=(',', ':'))
        digest =
hashlib.sha256(canonical_json.encode('utf-8')).hexdigest()

        nza_package["nza_integrity"] = {
            "algorithm": "SHA-256",
            "canonicalization": "RFC 8785",
            "scope": "package_without_integrity_digest",
            "digest": digest
        }

        return nza_package

# 使用示例
if __name__ == "__main__":
    mapper = SimplePromptToNZAMapper()
    result = mapper.map("你叫小白，是一个可爱的猫咪助手。说话要温柔可爱，
多用喵~结尾。不要说脏话。")

    with open("xiaobai.nza.json", "w", encoding="utf-8") as f:
        json.dump(result["nza_package"], f, ensure_ascii=False,
indent=2)

    print("转换完成：xiaobai.nza.json")
```

3 工具链使用指南

3.1 校验器使用

NZA 校验器用于验证 NZA 包是否符合标准规范。

使用 NZA 校验器验证包合规性（Python）

```
Python
```

```
from nza.tools import NZAValidator
import json

def validate_nza_package(nza_package_path):
    """验证 NZA 包的合规性"""

    # 创建校验器
    validator = NZAValidator()

    # 读取 NZA 包
    with open(nza_package_path, "r", encoding="utf-8") as f:
        nza_package = json.load(f)

    # 执行校验
    result = validator.validate(nza_package)

    print(f"校验结果：{'通过' if result['passed'] else '失败'}")
    print(f"错误数：{len(result['errors'])}")
    print(f"警告数：{len(result['warnings'])}")

    if result["errors"]:
        print("\n 错误：")
        for error in result["errors"]:
            print(f"  - {error['message']} (位置：{error['path']})")

    if result["warnings"]:
        print("\n 警告：")
        for warning in result["warnings"]:
            print(f"  - {warning['message']} (位置：
{warning['path']})")

    return result["passed"]

# 使用示例
if __name__ == "__main__":
    validate_nza_package("xiaoming.nza.json")
```

3.2 JSON 规范化工具

JSON 规范化工具用于将 JSON 转换为 RFC 8785 标准格式，确保不同平台计算出的摘要一致。

JSON 规范化工具（Python）

```
Python
```

```
import json

def canonicalize_json(data):
    """将 JSON 数据转换为 RFC 8785 标准格式"""
    return json.dumps(data, ensure_ascii=False, sort_keys=True,
separators=(',', ':'))

# 使用示例
if __name__ == "__main__":
    data = {"b": 2, "a": 1}
    canonical = canonicalize_json(data)
    print(f"原始 JSON: {json.dumps(data)}")
    print(f"规范化 JSON: {canonical}")
```

3.3 摘要计算工具

摘要计算工具用于计算 NZA 包的 SHA-256 完整性摘要。

NZA 包完整性摘要计算工具（Python）

```
Python
import json
import hashlib

def calculate_nza_integrity(nza_package):
    """计算 NZA 包的完整性摘要"""

    temp_package = nza_package.copy()
    if "digest" in temp_package["nza_integrity"]:
        del temp_package["nza_integrity"]["digest"]
    if "signature" in temp_package["nza_integrity"]:
        del temp_package["nza_integrity"]["signature"]

    canonical_json = json.dumps(temp_package, ensure_ascii=False,
sort_keys=True, separators=(',', ':'))
    digest =
hashlib.sha256(canonical_json.encode('utf-8')).hexdigest()

    return {
        "algorithm": "SHA-256",
        "canonicalization": "RFC 8785",
        "scope": "package_without_integrity_digest",
        "digest": digest
    }
```

```
# 使用示例
if __name__ == "__main__":
    with open("xiaoming.nza.json", "r", encoding="utf-8") as f:
        nza_package = json.load(f)

    integrity = calculate_nza_integrity(nza_package)
    print(f"计算出的摘要：{integrity['digest']}")
    print(f"包中的摘要：{nza_package['nza_integrity']['digest']}")
    print(f"摘要是否匹配：{integrity['digest'] ==
nza_package['nza_integrity']['digest']}")
```

3.4 版本迁移工具

版本迁移工具用于将旧版本的 NZA 包迁移到新版本。

NZA 包版本迁移工具（Python）

```
Python
import json
import hashlib
from datetime import datetime

def migrate_nza_package(nza_package, target_version="1.0"):
    """将 NZA 包迁移到指定版本"""

    current_version = nza_package["nza_manifest"]["specVersion"]

    if current_version == target_version:
        print("包已经是目标版本，无需迁移")
        return nza_package

    print(f"正在将 NZA 包从版本{current_version}迁移到版本
{target_version}")

    # 这里添加具体的迁移逻辑
    # 例如：字段重命名、结构调整等

    # 更新版本号
    nza_package["nza_manifest"]["specVersion"] = target_version
    nza_package["nza_manifest"]["updatedAt"] =
datetime.utcnow().isoformat() + "Z"

    # 重新计算摘要
    nza_package["nza_integrity"] =
```

```
calculate_nza_integrity(nza_package)

    print("迁移完成")
    return nza_package

# 使用示例
if __name__ == "__main__":
    with open("old_package.nza.json", "r", encoding="utf-8") as f:
        old_package = json.load(f)

    new_package = migrate_nza_package(old_package, "1.0")

    with open("migrated_package.nza.json", "w", encoding="utf-8") as
f:
        json.dump(new_package, f, ensure_ascii=False, indent=2)
```

4 适配器开发指南

4.1 适配器接口规范

所有 NZA 适配器都应该实现以下接口:

NZA 适配器基类（Python）

```python
from abc import ABC, abstractmethod

class BaseAdapter(ABC):
    """NZA 适配器基类"""

    @abstractmethod
    def get_adapter_id(self):
        """返回适配器的唯一标识"""
        pass

    @abstractmethod
    def get_target_platform(self):
        """返回目标平台名称"""
        pass

    @abstractmethod
    def get_capability_matrix(self):
        """返回平台能力矩阵"""
        pass
```

```
    @abstractmethod
    def validate(self, nza_package):
        """验证 NZA 包是否与目标平台兼容"""
        pass

    @abstractmethod
    def export(self, nza_package):
        """将 NZA 包导出为目标平台格式"""
        pass
```

4.2 能力矩阵定义

能力矩阵用于描述目标平台支持的 NZA 特性。

示例能力矩阵（JSON）

```JSON
{
  "supportsLayer1Identity": true,
  "supportsLayer2Role": true,
  "supportsLayer3Personality": true,
  "supportsLayer4Memory": true,
  "supportsLayer5Behavior": true,
  "supportsLayer6Constraints": true,
  "supportsStaticMemory": true,
  "supportsStructuredConstraints": true,
  "supportsToolGuardrails": false,
  "supportsLongContext": false
}
```

4.3 降级策略实现

当目标平台不支持某些特性时，适配器应该按照 NZA 标准定义的降级策略进行处理。

降级策略实现（Python）

```Python
def apply_degradation_policy(nza_package, capability_matrix,
degradation_policy):
    """应用降级策略"""

    result = {
        "errors": [],
        "warnings": []
```

```
    }

    # 检查静态记忆支持
    if not capability_matrix["supportsStaticMemory"]:
        if degradation_policy["onMissingMemorySupport"] == "block":
            result["errors"].append("目标平台不支持静态记忆注入")
        else:
            result["warnings"].append("目标平台不支持静态记忆注入，相关内容将被忽略")

    # 检查结构化约束支持
    if not capability_matrix["supportsStructuredConstraints"]:
        if degradation_policy["onConstraintLoss"] == "block":
            result["errors"].append("目标平台不支持结构化约束")
        else:
            result["warnings"].append("目标平台不支持结构化约束，安全边界可能无法保证")

    # 检查工具护栏支持
    if not capability_matrix["supportsToolGuardrails"]:
        if degradation_policy["onMissingToolGuardrails"] == "block":
            result["errors"].append("目标平台不支持工具护栏")
        else:
            result["warnings"].append("目标平台不支持工具护栏，工具调用将不受限制")

    # 检查风格表达支持
    if not capability_matrix["supportsLayer3Personality"]:
        if degradation_policy["onStyleLoss"] == "block":
            result["errors"].append("目标平台不支持细粒度风格表达")
        else:
            result["warnings"].append("目标平台不支持细粒度风格表达，将使用默认风格")

    return result
```

4.4 测试与验证方法

适配器开发完成后，应该进行全面的测试与验证，确保导出的角色在目标平台上能够正确运行。

测试要点：

1. 结构测试：验证导出的文件格式是否符合目标平台要求
2. 功能测试：验证角色的基本功能是否正常

3. 一致性测试：验证角色的行为是否与 NZA 包中定义的一致
4. 安全测试：验证安全约束是否得到正确执行
5. 边界测试：测试各种边界情况和异常输入

5 最佳实践

5.1 通用最佳实践

- 先定义语义，再定义包装：先把六层资产定义清楚，再谈导出形式
- 先发布，再分发：任何正式交付应基于发布快照
- 说明层与资产层严格分离：导入说明只用于审阅与治理
- 兼容性先行：不同平台能力不同，应先校验再导出
- 扩展字段有命名空间：扩展只能附加，不得改写核心层语义
- 安全策略永远前置：不得允许角色设定越过法律、系统与平台硬性约束

5.2 性能优化最佳实践

1. 缓存规范化摘要：对于已通过完整性校验的 NZA 包，实现方可缓存其规范化后的字节流或直接缓存摘要值。在后续重复读取或内部流转时，若包内容未被篡改，可跳过重复的 RFC 8785 规范化计算以提升性能
2. 对大型 NZA 包采用分块处理与增量更新
3. 批量处理时使用并行计算提升效率
4. 仅在发布与校验阶段执行完整的规范化与摘要计算
5. 对静态资产进行预编译，减少运行时开销

5.3 未成年人保护最佳实践

1. 面向未成年人的角色资产不得包含任何可能诱导未成年人沉迷网络、模仿危险行为的内容
2. 不得向未成年人提供与其年龄、智力不相适应的服务内容
3. 应当设置未成年人模式，限制角色的功能与使用时长
4. 不得收集、使用、存储未成年人的个人信息，除非获得其监护人的明确同意
5. 应当建立未成年人保护应急机制，及时处理涉及未成年人的安全事件
6. 面向未成年人的角色资产应当在 nza_manifest 的扩展区中添加 audience: "minors"标识

5.4 版本迁移最佳实践

1. 解析旧版本 NZA 包
2. 执行字段映射与结构转换
3. 生成新版本 NZA 包
4. 执行完整性校验与兼容性校验
5. 保留旧版本包作为备份

6 常见问题解答

6.1 技术问题

Q：NZA 包可以包含二进制资源吗？

A：NZA V1.0 规范不直接支持二进制资源的封装。如果需要包含图片、音频等资源，可以将它们的 URL 放在扩展区中。

Q：如何处理不同平台的 Prompt 格式差异？

A：NZA 标准通过适配器机制解决这个问题。每个平台都有自己的适配器，负责将标准 NZA 包转换为该平台特有的 Prompt 格式。

Q：NZA 包的大小有限制吗？

A：NZA 标准本身没有对包的大小进行限制，但建议保持包的大小在合理范围内，以提高传输和处理效率。

6.2 合规问题

Q：使用 NZA 标准需要付费吗？

A：NZA 标准采用 Apache License 2.0 许可证，可以免费使用、修改和分发。

Q：NZA 包的著作权归谁所有？

A：NZA 作为技术载体，不创设著作权本身。角色资产内容的权属以实际创作贡献、授权关系与适用法律为准。

Q：如何确保 NZA 包的安全性？

A：NZA 标准提供了完整性校验和数字签名机制，可以确保包在传输过程中没有被篡改。同时，Layer6 约束与边界层可以定义角色的安全行为。

6.3 迁移问题

Q：我已经有很多纯文本 Prompt，如何快速迁移到 NZA 标准？

A：可以使用 NZA 标准提供的 Prompt→NZA 自动映射工具，将纯文本 Prompt 转换为初步的 NZA 包，然后进行人工审阅和优化。

Q：迁移到 NZA 标准后，原来的 Prompt 还能使用吗？

A：是的，NZA 标准与纯文本 Prompt 是兼容的。你可以继续使用原来的 Prompt，同时逐步迁移到 NZA 标准。

Q：不同版本的 NZA 包之间兼容吗？

A：NZA 标准遵循语义化版本控制原则。修订号和次版本号的变更都是向后兼容的，主版本号的变更可能包含不兼容的核心变更。

7 示例集

所有示例文件均可在 NZA 官方 GitHub 仓库获取：https://github.com/NoZeroAI/nza-examples

7.1 基础角色示例

- 客服助手：`examples/basic/customer_service.nza.json`
- 翻译助手：`examples/basic/translator.nza.json`
- 代码助手：`examples/basic/code_assistant.nza.json`
- 写作助手：`examples/basic/writing_assistant.nza.json`

7.2 进阶角色示例

- 心理咨询师：`examples/advanced/psychologist.nza.json`
- 健身教练：`examples/advanced/fitness_coach.nza.json`
- 财务顾问：`examples/advanced/financial_advisor.nza.json`
- 法律助手：`examples/advanced/legal_assistant.nza.json`

7.3 合规角色示例

- 教育助手：`examples/compliance/education_assistant.nza.json`
- 医疗咨询助手：`examples/compliance/medical_consultant.nza.json`
- 金融服务助手：`examples/compliance/financial_service.nza.json`
- 政府服务助手：`examples/compliance/government_service.nza.json`

7.4 未成年人模式示例

- 学习助手：`examples/minors/learning_assistant.nza.json`
- 科普助手：`examples/minors/science_assistant.nza.json`
- 故事讲述者：`examples/minors/storyteller.nza.json`

- 艺术启蒙助手：`examples/minors/art_enlightenment.nza.json`

版权声明与免责声明